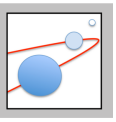
Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Prof. Dr. –Ing. Hakan Kayal
Professorship of Space Technology
Chair of Computer Science VIII



Master Thesis

Development and Implementation of an Image-Processing-Based Horizon Sensor for Sounding Rockets

submitted by

Jochen Barf

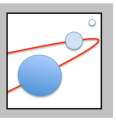
to the Department of Mathematics and Computer Science
in Partial Fulfilment of the Requirements of the Degree of Master of Science
with the Major of Space Science and Technology

Co-funded by the
Erasmus+ Programme
of the European Union

Deutsches Zentrum
für Luft- und Raumfahrt
German Aerospace Center

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Prof. Dr. –Ing. Hakan Kayal
Professorship of Space Technology
Chair of Computer Science VIII



Master Thesis

Development and Implementation of an Image-Processing-Based Horizon Sensor for Sounding Rockets

submitted by

Jochen Barf[1]
born on 18th November 1990 in Schweinfurt

to the Department of Mathematics and Computer Science
in Partial Fulfilment of the Requirements of the Degree of Master of Science
with the Major of Space Science and Technology

Created at:
Chair of Aerospace Information Technology
Professorship of Space Technology
Julius–Maximilians–University of Würzburg

Supervisors:
Dr. Benjamin Braun (DLR)
Markus Wittkamp (DLR)

Examiners:
Prof. Dr.-Ing. Hakan Kayal (JMUW)
Assoc. Prof. Dr. Thomas Kuhn (LTU)

Submission Date:
1st July 2017

[1] jochen.barf@stud-mail.uni-wuerzburg.de

Declaration of Authorship

I declare that the work presented here is, to the best of my knowledge and belief, original and the result of my own investigations, except as acknowledged, and has not been submitted, either in part or whole, for a degree at this or any other university. Formulations and ideas taken from other sources are cited as such. This work has not been published before.

Würzburg, 1st July 2017

(Jochen Barf)

Acknowledgements

I would like to thank Dr. Benjamin Braun and Markus Wittkamp for their outstanding support and encouragement throughout all phases of the thesis. I would also like to thank Prof. Dr. Hakan Kayal and Prof. Dr. Thomas Kuhn for their advice in many different aspects.

Abstract

In this thesis a new flexible and robust horizon sensor software for sounding rockets is presented. It operates on image data in the visible spectrum of light and provides two axis attitude information. After a short analysis of the current available technology, the novel algorithm is explained in detail and the new mathematical concept is thoroughly defined. It features a very specialised and optimized edge detection, outlier detection, fisheye distortion compensation and a plausibility check. Moreover, it can cope with strong disturbances introduced by the Sun and inhomogeneities of the reflectance of the Earth while maintaining a linear runtime in the number of pixels. No a priori knowledge apart from the altitude is required by the algorithm. With simulations using Google Earth Pro, the major influential factors to the accuracy and their impact are determined. The software in `C++` was implemented for the DSP Blackfin 561 and the RODOS realtime kernel. A tool that simulates an ethernet camera was developed in order to validate the functionality of the software on the DSP. To determine the intrinsic parameters of an optical system, a program that analyses images of chessboard patterns was implemented using OpenCV. The evaluation determined an accuracy better than 1° in the simulation with a sample time between 0.2s and 1s, depending on the set of parameters. The minimum altitude of 25km and the rate of false positive detections of 5% were determined with real footage of a sounding rocket flight from GoPro cameras.

Contents

CHAPTER 1

Introduction

Horizon sensors (HSs) are reliable two axis attitude sensors used on satellites to determine the attitude with respect to Earth. In some literature, horizon sensors are therefore referred to as Earth sensors. In this thesis, however, the more suitable term horizon sensor is consistently used. HSs were primary sensors used from low Earth orbit (LEO) to geostationary orbit (GEO) but their usage is progressively decreasing for the last 10 years in favour of the more accurate star sensors. That is the reason why the number of products on the market is relatively small [Mazzini, 2016].

1.1 State-of-the-Art of Horizon Sensors

The albedo of the Earth lies mostly in the visible spectrum but its intensity varies wildly with the reflecting surface (water, ice, forest, snow, soil). The albedo in the infrared (IR) spectrum, especially between 14-16μm (CO_2 band), has a uniform energy distribution irrespective of day and night. Therefore, most traditional HS use a bolometer [Ley et al., 2009]. A bolometer is a very sensitive resistance thermometer, or thermistor, used to detect infrared radiation.
HS are mostly used for Earth related missions with medium pointing accuracy. In LEO the Earth occupies about 40% of the field of view (FOV) [Ley et al., 2009] but since HS are disturbed by the Sun or Moon they have to be disabled for the time those disturbances are in the FOV [Uhlig et al., 2015]. In GEO the Earth occupies a solid angle of about 17.4° whereas the sun only occupies 0.5° [Berlin, 1988]. Hence, using the Earth as reference is a reasonable choice. By detecting the infrared disk of the Earth, HSs determine the nadir vector which corresponds to a pitch and roll angle measurement of the spacecraft with respect to the Earth. The yaw axis, the rotation around the nadir vector, is not observable [de Ruiter et al., 2013]. Traditional HS can

be classified in two types: scanning and static HS. Figure 1.1 illustrates the working principle.

1.1.1 Scanning Horizon Sensors

Scanning HS are either deployed on spin-stabilized spacecraft or have a rotating optical head [Macdonald and Badescu, 2014]. The IR sensor detects if the the Earth is in the instantaneous FOV. Due to the rotation, the Earth enters and leaves the instantaneous FOV once every revolution. The time at which the signal is acquired is

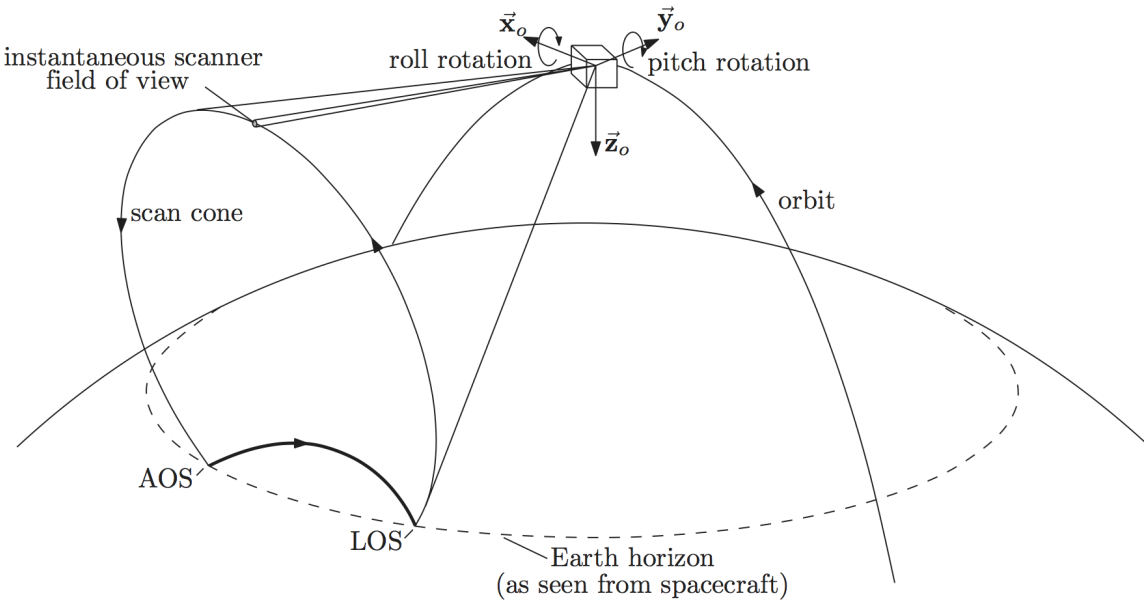
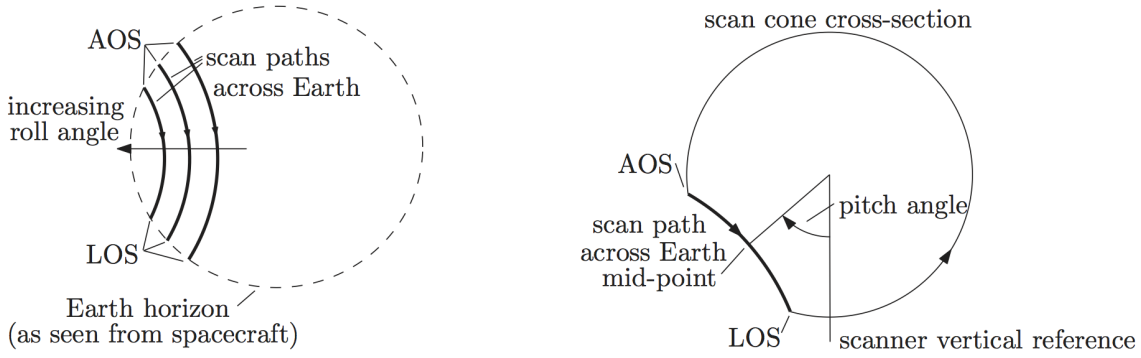


Figure 1.1: Working Principle of a scanning HS [cf. de Ruiter et al., 2013, Fig. 26.5]

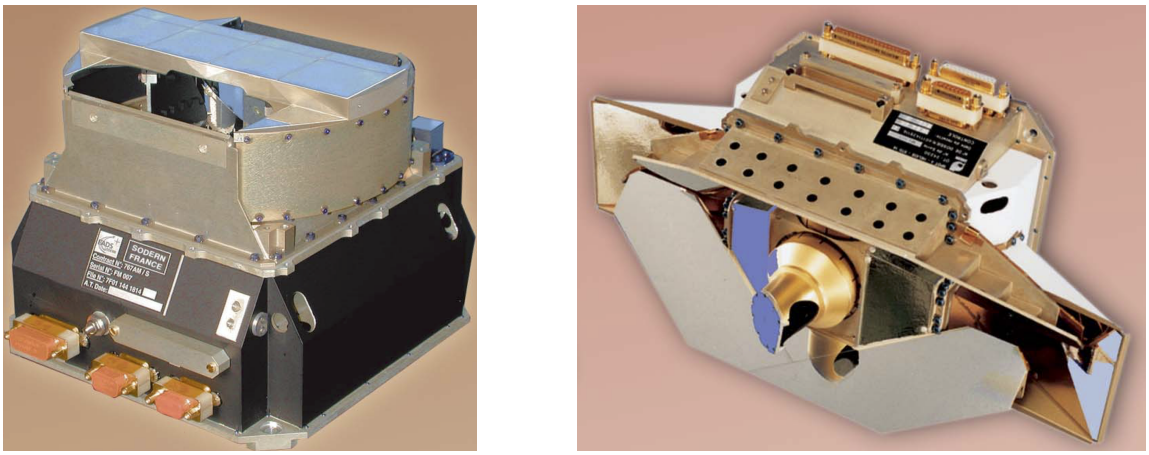
called acquisition of signal (AOS) and the time at which the signal is lost is called loss of signal (LOS). As illustrated in Figure 1.2a, the time difference between AOS and LOS is a measure of the roll angle of the spacecraft. The position of the passthrough can be translated into a pitch angle as illustrated in Figure 1.2b. For this method to work, it is important, that the altitude is known and that the spacecraft's body rate is small compared to the rotation rate of the sensor [de Ruiter et al., 2013]. Typical examples of such sensors are the STD-15 and STD-16 manufactured by EADS Sodern. A photo of both sensors can be seen in Figure 1.3. The STD-15 was developed for GEO satellites but is operable at altitudes between 15000km and 140000km with an accuracy of $\pm 0.05°$ (3σ) and a maximum range of $\pm 15.6°$ in pitch or $\pm 14.5°$ roll. The STD-16 was designed for LEOs from 300km to 6000km. Within a maximum range of $\pm 17°$ in pitch or $\pm 33°$ roll it reaches an accuracy of $\pm 0.1°$ (3σ) [EADS Sodern].



(a) Roll Angle Determination

(b) Pitch Angle Determination

Figure 1.2: Roll and Pitch Angle Determination of Scanning HS [cf. de Ruiter et al., 2013, Fig. 26.6]



(a) STD-15

(b) STD-16

Figure 1.3: Scanning HS Examples [cf. EADS Sodern]

1.1.2 Static Horizon Sensors

Traditional static HSs however require the entire Earth to be inside the FOV and are therefore used in higher orbits [Macdonald and Badescu, 2014]. This type of HSs are also less suitable for elliptical orbits [Macdonald and Badescu, 2014] because the FOV is optimized for one specific altitude [Pisacane, 2005]. The basic setup of typical static HS is a set of concentrically arranged IR sensors as illustrated in Figure 1.4. The Earth, as seen by the sensor, appears as large disk and the signal from each IR sensor is proportional to the fraction contained in its FOV [de Ruiter et al., 2013]. These measurements can be directly translated into roll and pitch deviations. An example of such a sensor is the IRES-C manufactured by Sellex Galileo (cf. Fig. 1.5). The

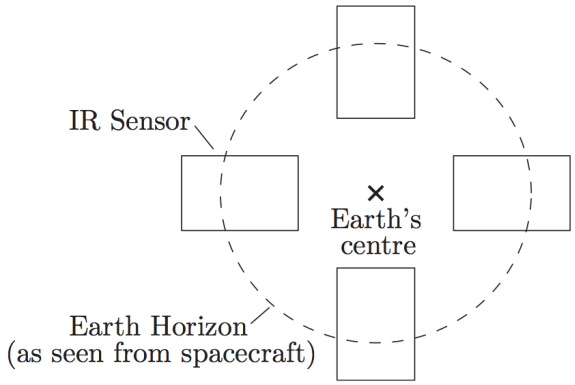


Figure 1.4: Working Principle of a static HS [cf. de Ruiter et al., 2013, fig. 26.4]

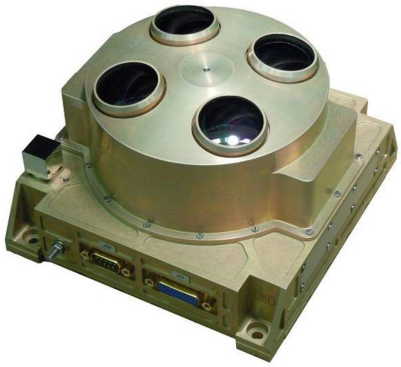


Figure 1.5: IRES-C [cf. Sellex Galileo]

IRES-C, which stands for Infrared Earth Sensor Coarse, comes in two configurations, one for LEO and one for GEO. The LEO configuration provides attitude information with an accuracy of $\pm 1.4°(3\sigma)$ in a FOV of $\pm 86°$ within an altitude range of 500km to 2000km. The GEO configuration operates between 30000km and 42000km with an accuracy of $\pm 0.5°$ (3σ) in a FOV of $\pm 12°$ [Sellex Galileo].

1.1.3 New Generation Horizon Sensors

The reason the sensors above are referred to as "traditional" HSs is that there has been an accelerating effort in the community towards horizon sensors with two-dimensional optical sensor arrays primarily in the visible spectrum. This development might be tied to the recent advances in complementary metal-oxide-semiconductor (CMOS) technology and their drop in costs. However, there was only one commercially available product found during this work: the CubeSense sensor manufactured by CubeSpace. According to Cube Space [2016] the device features a 1024×1024 CMOS

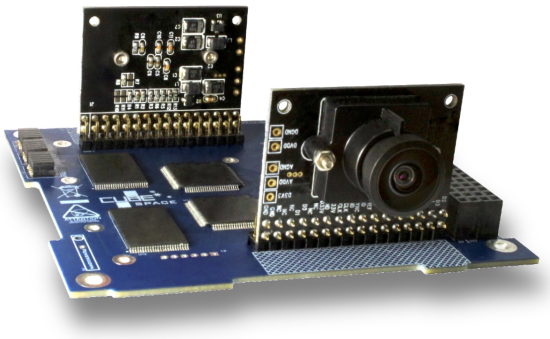


Figure 1.6: CubeSense [cf. Cube Space, 2016]

sensor sensitive to the visible spectrum. It is designed for LEOs and can provide attitude data with an accuracy of $\pm 0.2^\circ$ inside a 150° FOV with a sample rate of 1Hz. Unfortunately, no further information about the functional principle is available, not even after a personally requesting for more information. A paper that seems to be related is [Bahar et al., 2006] but this cannot be confirmed due to the withheld information. Bahar et al. describe a device that consists of a CMOS sensor with 1024×1024 pixels, operates in the visible spectrum and provides data with a frequency of 1Hz. The design for LEOs has an accuracy of 0.1° within a FOV of 90°. The corresponding paper states, that the algorithm includes a Sobel edge detection with a subsequent circle fit.

Meller et al. [2000] introduced a sensor system for LEOs with multiple CMOS devices with 512×512 pixels sensitive to the visible spectrum. Each device has a FOV of 67° and provides attitude data with a data rate of 2Hz. With a threshold filter applied to a small number of scan lines a small number of horizon points are determined and fed into a Least-Squares circle fit.

Rensburg [2008] proposed a sensor sensitive to the IR spectrum between $6.5\mu m$ and $20\mu m$ wavelength. The device, designed for LEO satellites, has a 32×31 pixel image sensor and provides attitude data with $\pm 0.24^\circ$ (3σ) pitch and $\pm 0.59^\circ$ (3σ) roll accuracy within an FOV of 35° every 52ms. The applied algorithm consists mainly of a sub-pixel Canny edge detection with a subsequent line fit using the least-squares method. All of the sensors described above require a priori knowledge about the location of the horizon in the image and some are only operable in certain altitudes.

1.2 Motivation

The Mobile Rocket Base (MORABA) is a department of the institute of Space Operations and Astronaut Training within the German Aerospace Center (DLR). The MORABA is, apart from the U.S. National Aeronautics and Space Administration (NASA), the only institution in the western world with the ability and mobile infrastructure to prepare and conduct high altitude science missions across the globe. High altitude rockets (also known as sounding rockets) provide a micro gravity phase between 3 and 15 minutes and can reach altitudes up to 800km. Many of the available vehicles are spin stabilized and rotate with frequencies up to 4Hz [MORABA]. After burnout the spin is almost neutralized and the rockets are stabilized with the onboard attitude determination and control system (ADCS) or remain uncontrolled depending on the experiment requirements. In the latter case it is often sufficient to reconstruct the attitude in post processing after the flight but in the former case, the attitude must be determined on board in real time. This is currently achieved with the Digital Miniature Attitude Reference System (DMARS) manufactured by

Inertial Science Inc. It is an inertial measurement unit (IMU) with multiple gyroscopes and accelerometers particularly designed for vehicles with high roll rates up to 22 revolutions per second (RPS). It provides three axis attitude information with an accuracy of $\pm 0.3^\circ$ in azimuth and $\pm 0.1^\circ$ in level axis. With 11cm in diameter, a height of 16.5cm and a mass of 2.5kg, not including the heatsink, the key word "miniature" is not contemporary. Also its power consumption of 22W is a major drawback. Moreover, the maintenance requires company personnel and the device is subject to the International Traffic in Arms Regulations (ITAR) [Inertial Science Inc., 1999].

The department Space Flight Technology (RFT) of the German Space Operation Center (GSOC) therefore aims to develop a smaller, cheaper and lighter alternative with less power consumption and similar accuracy compared to the currently used DMARS-R platform. The idea is to use cheap commercial off-the-shelf (COTS) components and take advantage of a powerful onboard computer. The problem with low cost COTS IMUs is a lower accuracy especially in highly dynamical regions. Since the IMU data must be integrated to determine attitude information, the errors of all previous samples contribute to the current attitude error. This creates a drift in the attitude measurement that can increase the errors in the order of several degrees within a few seconds. Although this drift can be reduced with data fusion algorithms like a Kalman filter, a drift-free attitude determination with only COTS inertial sensors is rather impossible. Therefore, sensors that determine the attitude independent from previous samples and thus do not accumulate an error are required. These sensors are called reference sensors because their attitude data is given with respect to a reference independent from the inertial frame. Such sensors are e.g. star trackers, sun sensors and horizon sensors. Of course their attitude information is error-prone but its error is restricted to a single measurement and independent from all other measurement errors. Reference sensors, however, usually have a much lower data rate compared to inertial sensors. With a data fusion algorithm that compensates the drift with updates acquired from a reference sensor, the advantages of both worlds can be combined. The result is an accurate and drift-free attitude information with high sample rates.

Star sensors cannot be operated in highly dynamical regions or if the sun is in the FOV and thus are unapt for sounding rockets. The Earth horizon is a very dominant feature in the environment of LEOs or suborbital flights and is therefore a good target for reference sensing. Nevertheless, there is currently no HS for sounding rockets available although the prospects of such a sensor are very promising. Therefore, this thesis aims to take a big step towards a functioning horizon sensor based on image processing techniques for sounding rockets using COTS components. Such a sensor may also be of significance for cubesats since the variety of HSs for this market is very limited. Therefore, a highly flexible and robust design that can be adapted to many configurations is preferred. The proposed sensor would have a high emphasis

on the software and the underlying image processing techniques and mathematical concepts instead of highly specialized hardware. This approach would have a variety of advantages: Many spacecraft, regardless if orbital or suborbital, already carry cameras and have continuously increasing onboard computing capabilities. The proposed design would be able to make use of the preexisting hardware and would thus not further strain the mass budget. Software in its nature is very flexible and can, in contrary to hardware, be updated while in orbit. Of course the usage of COTS components decreases the costs. If the attitude of a spacecraft does not need to be available in realtime but only after post processing on ground it is sufficient to only fly a camera that records a video onboard and perform the attitude determination later on a desktop computer.

A robust design of the software would not need any a priori information about the location of the horizon in the image, would be able to cope with disturbances like the sun or reflections and would be independent of the spacecraft's rate of revolution. This would not require any special mounting angle of the camera on the vehicle as long as the the horizon is in the field of view. Since the Earth is the largest object in the environment of the spacecraft it is very likely to be captured by the camera even in very high-dynamic regions. A sensor in the visible spectrum of light is preferred due to higher resolutions, lower costs, and higher variety of available components. However, a flexible software should be able to work with both visible and IR image data. A fisheye lens with a high FOV would increase the probability that the horizon is in the image.

1.3 Related Work

The REXUS/BEXUS programme offers free flight tickets on high altitude balloons (BEXUS) and sounding rockets (REXUS) for student experiments. Student teams of typically four to six students design, build, fly and evaluate their experiment within a two-year project cycle. In the year 2014 the REXUS team Horizon Acquisition Experiment (HORACE) of six undergraduate students from the University of Würzburg launched an experimental prototype of a HS on the rocket REXUS 16. Their goal was to proof the concept of detecting the curvature of the Earth in images of ordinary cameras operating in the visible spectrum of light [Rapp et al., 2014]. Due to a malfunction of the cameras the ultimate proof of concept was not possible. The algorithm itself however showed very promising results with footage from a previous flight. It was implemented in `C++` using the open source computer vision library OpenCV and ran on top of Arch Linux on a micro computer Pico-ITX MIO-2260 with an Intel Atom CPU [Barf et al., 2015]. In an effort to push this idea to the next logical step, the algorithm was reimplemented in the context of a bachelor thesis by Barf without using any source code besides the standard `C++` libraries. The follow up

REXUS team of HORACE, Position-Vector Acquisition Through Horizon Observation System (PATHOS) from the very same professorship of Prof. Dr. Hakan Kayal seized on the idea of HORACE and aimed to proof the concept while miniaturising the hardware. PATHOS used the implementation developed in [Barf, 2014], optimised the sample time and ran it on a OVERO board manufactured by Gumstix. In 2015 PATHOS flew their experiment on board the REXUS 20 and finally proved the concept [Wagner et al., 2016].

1.4 Aims and Objectives

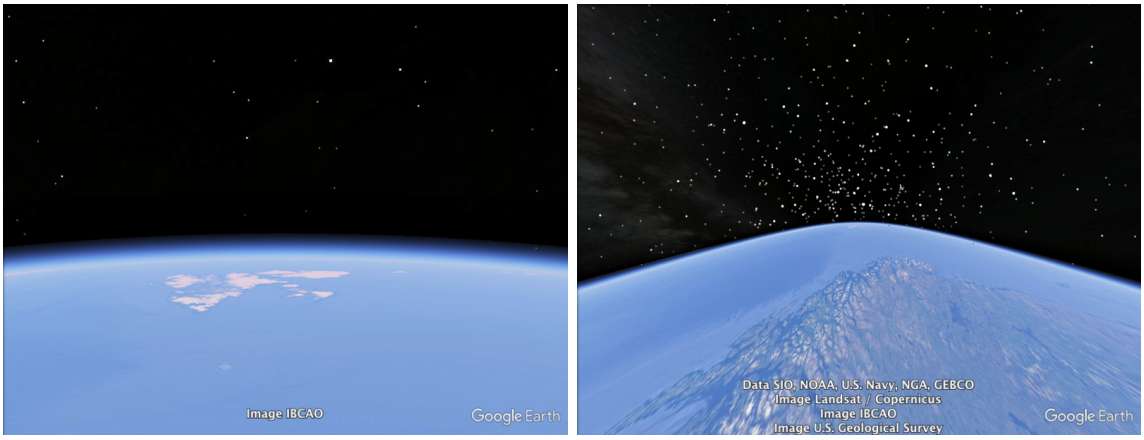
In order to achieve a new ADCS for sounding rockets (cf. Sec. 1.2), this work aims to create a reference sensor by building up on the work by HORACE, Barf and PATHOS (cf. Sec. 1.3). The primary aim of this thesis is an operable implementation of a horizon sensor software on the target platform that, provided with image data of the Earth horizon, is able to calculate two-axes attitude information of the camera with respect to Earth. The secondary aim is to determine the accuracy, sample time and robustness of such a system and their major influential factors.

The target platform is a preexisting embedded system described in detail in Section 3.1.2. Test environments to validate the implementation on the target platform and to analyze the accuracy, sample time and robustness are to be developed. The validation of the implementation on the embedded system is to be done with simulated data only. Neither the selection of a camera nor the implementation of its driver is part of this thesis. The algorithm and its implementation described in Section 1.3 are to be optimzed regarding robustness, sample time and accuracy. The sample time should not exceed 1s and the systematical error of the attitude should be below 1°. The robustness is to be tested with real footage if available. A rate of 10% false negatives should not be exceeded. The algorithm should be as flexible and adaptable as possible. It should not require any a priori knowledge of the location of the horizon in the image. It should be operable at altitudes typical for sounding rockets and cubesats. The altitude information may be used as a priori information in the algorithm. The algorithm and its implementation are to be modified in order to enable the usage of fisheye lenses with high FOV.

1.5 Emphasis of Work

During the development of the accuracy evaluation environment (cf. Sec. 4.3) it was found that an assumption, the algorithm described in Section 1.3 is based on, is

incorrect. The same assumption is made by Bahar et al., Meller et al. and Rensburg. It was consistently assumed that the projected curve of the horizon of the Earth in the image is always a circle. Bahar et al., Meller et al. and Barf determined the center of the circle to calculate the attitude. Rensburg even used a straight line fit and determined the attitude from the position and inclination of the estimated straight line. Crisman [2016] used machine learning techniques to determine if a straight line, circle or parabola best fitted the observed curve. The concept derived in Section 2.5 suggests that neither of them is correct in most of the cases. The most likely curve of the projection is either an ellipse or a hyperbola, depending on the altitude as analysed in Section 4.1. Only in very special cases a circle or a parabola is possible, but these cases are so special that they are basically not observable. This statement is very counterintuitive but becomes noticeable when looking at a simulated view of the Earth horizon by Google Earth Pro (GEP) at a high and a low FOV as seen in Figure 1.7. In the left picture with a FOV of 40° only a small portion of the curve is



(a) FOV=40° (b) FOV=150°

Figure 1.7: Google Earth Simulated View with High and Low FOV

visible. That picture does not raise any concern that this is not a portion of a circle. That this is a misconception is thoroughly explained in Section 2.5 and illustrated in Figure 1.7b. It shows the same view at the same altitude but with a much higher FOV of 150°, therefore a greater portion of the projected curve is visible. In contrast to the image on the left, this image does not leave any doubt that the projected curve is not a circle. The theory in Section 2.5 states that the curve of the horizon as seen in Figure 1.7b is in fact one branch of an hyperbola. An algorithm that uses a circle fit or even a fit for a straight line produces considerable systematic errors.

In order to achieve the best possible result one must regard the projected curve as non-degraded general conic section. This increases the complexity of the algorithm

significantly. Nevertheless, the emphasis of this thesis was directed towards the development of the mathematical concept to model the projection as accurately as possible and to create an algorithm that uses this model to estimate the attitude. The developed algorithm is presented in the next chapter.

CHAPTER 2

The Algorithm

Cormen et al. state: "Informally, an algorithm is any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output. An algorithm is thus a sequence of computational steps that transform the input into the output." In this application the input is a picture, the output is attitude information. The definition of the steps is the subject of this section.

2.1 Basic Idea

The idea of this algorithm is to find the curvature of the central body's projected horizon in the image and to draw a conclusion from that curvature about the attitude of the camera with respect to the central body. In order to do so, the algorithm follows six logical steps: threshold filter, edge detection, undistortion, outlier detection, vector calculation and plausibility check.

First, it is determined which part of the image is occupied by the central body. This is done by the threshold filter which converts the image to a binary image based on the brightness of each pixel. The horizon is located at one of the borders between bright and dark pixels. In the edge detection step those pixels, that are at such a border, are extracted. Unfortunately, real optical systems introduce distortions to the image which must be compensated for. This is taken care of in the undistortion step. Due to the sun or inhomogeneities of the albedo of the central body, the algorithm is faced with heavy disturbances which can cause a massive amount of outliers in the measurement. Therefore, the outliers are detected and removed. Up to this step, there are only pixel coordinates that are undistorted and cleaned of outliers but no

attitude information. In the next step, the nadir vector is calculated by taking all of the selected pixel coordinates into account. Finally, the plausibility and quality of the result is assessed.

2.2 Further Development of Bachelor Thesis

The algorithm is greatly based on the algorithm introduced in the bachelor thesis *Development and Implementation of an Horizon Sensing Algorithm based on Image Processing Technologies* by Barf [2014]. In that work, the projection of the horizon was assumed to be a circle for any attitude. However, during the work on this thesis it was discovered, that this assumption is fundamentally wrong and creates a significant error in the measurement. Therefore, the algorithm's behaviour of interpreting the data had to be completely redesigned in order to meet the requirements (cf. Sec. 1.4). This included the derivation of the projection model explained in Section 2.5 and the development of a new estimation algorithm explained in Section 2.9. Especially the latter posed a much more complicated task because the data had to be fitted to a general conic section instead of a circle which is a much more complex process. Also the verification process in [Barf, 2014] was based on that assumption and had to be replaced. The radius of the detected circle in the image was compared to the expected one and the detection result discarded if the error exceeded a certain limit. The expected radius was calculated with a formula which is valid if the projection is a circle but happens to be the latus-rectum (cf. Sec. 2.5) of a general conic section in all other cases. The algorithm worked because the limits were set generously high and thus included the systematic error but the derived attitude information would have had a massive systematic error.

Apart from those corrections the algorithm was also optimized regarding computational intensity and robustness. In the previous version the entire image was converted to a binary image before the edge detection was started. As stated by Wagner et al. [2016], this is an unnecessary high amount of operations and can be drastically reduced by only converting those pixels that are actually used by the edge detection. In addition and also remarked by Wagner et al. [2016], it is unnecessary for the topological search algorithm in the edge detection step (cf. Sec. 2.6) to search the entire image for edge pixels. Wagner et al. instead proposed to only search the image borders and thus reduce the runtime from $\mathcal{O}(wh)$ to $\mathcal{O}(w + h)$ operations where w is the image width and h is the image height. This method however only works reliably if the projected curve always intersects the image border which is true in lower altitudes as can be read from the graph in Figure 4.3. Since this algorithm should be as flexible as possible, a self-adapting procedure is introduced and explained in Section 2.6. In the version of the algorithm initially proposed by Barf [2014] only the longest line (most points) found by the edge detection was considered. This is a very

practical solution but does not exploit all given information. In the new version the algorithm calculates the attitude for all found lines and chooses the one that fits the best. Moreover, the method of the outlier detection was changed. In the last version, outliers were detected by dividing the horizon line multiple times and calculating the vector for each partial segment. This method was replaced by the well known random sample consensus (RANSAC) algorithm by Fischler and Bolles.

2.3 Conventions and Nomenclature

A vector is indicated by an arrow on top of the symbol and always has curved brackets. For example the vector with n rows can be written as

$$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{pmatrix}. \tag{2.1}$$

For geometrical vectors with up to three components the numbering is replaced by the small letters x,y and z. The coordinate system the vectors are defined in is indicated with a vertical line with the identifier of the system as subscript. A vector in the three dimensional coordinate system s is for example written as

$$\vec{v}|_s = \begin{pmatrix} v_x \\ v_y \\ v_z, \end{pmatrix}. \tag{2.2}$$

The vectors $\vec{x}_s$, $\vec{y}_s$ and $\vec{z}_s$ are unit vectors in x, y and z direction of the coordinate system s. The scalar product is indicated with a small dot like

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i \cdot v_i \tag{2.3}$$

and the norm of a vector $\vec{v}$ is written as

$$||\vec{v}|| = \sqrt{\vec{v} \cdot \vec{v}} = \sqrt{\vec{v}^{\,2}} \tag{2.4}$$

where the dot product of a vector $\vec{v}$ with itself may be noted as $\vec{v}^{\,2}$ A matrix R with n rows and m columns is written with square brackets like

$$R = \begin{bmatrix} r_{11} & \dots & r_{1m} \\ \vdots & \ddots & \vdots \\ r_{n1} & \dots & r_{nm} \end{bmatrix}. \tag{2.5}$$

A transformation matrix R that transforms a vector $\vec{v}|_{s_1}$ into the system s_2 is written as

$$\vec{v}|_{s_2} = R|_{s_2}^{s_1} \cdot \vec{v}|_{s_1}. \tag{2.6}$$

A transformation $R|_{s_2}^{s_1}$ might be composed of multiple rotations around certain axes, with certain angles in a certain sequence. $R_{k^i}(\theta)$ depicts the ith rotation around the k-axis with angle θ. For example:

$$R|_{s_2}^{s_1} = R_x(\theta_x)R_{z'}(\theta_z) \tag{2.7}$$

means the system s_2 is first rotated around the x-axis by the angle θ_x and then rotated around the new z'-axis by the angle θ_z with respect to the system s_1.

2.4 Coordinate Systems

For convenience four different cartesian coordinate systems are used. Figure 2.1 illustrates all of them in an example configuration.

Please note, that the physical location of the image plane is at $z|_c = -f$ where f is the focal length of the optical system. However, the illustration shown in Figure 2.1 is an equivalent geometrical representation where the image plane is $z|_c = f$ which happens to be a much more intuitive description.

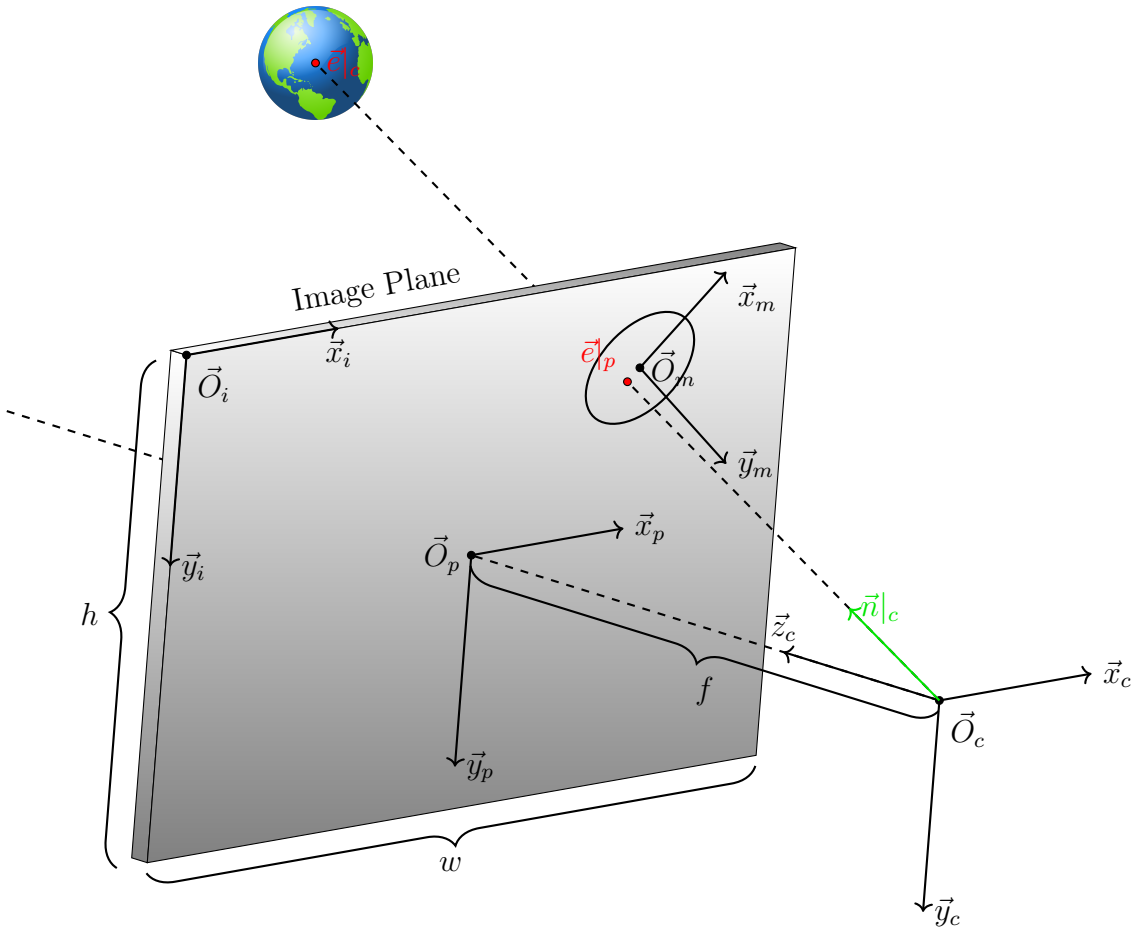


Figure 2.1: Definitions of the Coordinate Systems: Camera (c), Image (i), Principal Point (p) Conic Center (m)

2.4.1 Camera Coordinate System

The camera coordinate system has its origin in the focal point of the optical system and the $\vec{z}_c$ axis coincides with its principal axis. It is indicated with subscript c. The $\vec{x}_c$ and $\vec{y}_c$ axis complete a right handed orthogonal coordinate system where the $\vec{x}_c$ points to the right and $\vec{y}_c$ to the bottom of the camera as illustrated in Figure 2.2.

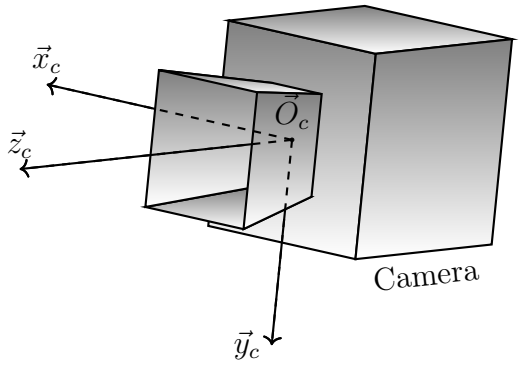


Figure 2.2: Camera Coordinate System

2.4.2 Principal Point Coordinate System

In contrast to the camera system the principal point coordinate system, indicated with subscript p, is a two-dimensional, right-handed, orthogonal coordinate system. Its origin is the principal point (center of the image) which is the intersection of the $\vec{z}_c$ axis and the image plane. The image plane is parallel to the $\vec{x}_c\vec{y}_c$-plane at distance f in $\vec{z}_c$ direction. The transformation

$$\vec{p}|_p = \frac{f}{p_z|_c} \begin{pmatrix} p_x|_c \\ p_y|_c \end{pmatrix} \tag{2.8}$$

is a shift and a scaling according to the principle of perspective of the pinhole camera model. This transformation is not bidirectional because of the loss of depth information.

2.4.3 Image Coordinate System

The image coordinate system is the system the image data is actually given in and is here indicated with subscript i. Its origin is in the upper left corner, its $\vec{x}_i$ axis points to the right and its $\vec{y}_i$ axis points down. Like the principal point system and unlike the camera system, the image system is a two-dimensional, right-handed, orthogonal system. The coordinate transformation between the p- and i-system is a simple shift

$$\vec{p}\,|_i = \vec{p}\,|_p + \begin{pmatrix} w/2 \\ h/2 \end{pmatrix} \tag{2.9}$$

where w is the image width and h the image height.

2.4.4 Conic Center Coordinate System

The conic center coordinate system is a special coordinate system because it is unique for every observation. Its origin is in the center of the conic section that is the projected curve of the horizon. The $\vec{x}_m$ and $\vec{y}_m$ axes lie in the image plane. A detailed derivation of that curve is given in Section 2.5. The transformation is a rotation with angle ρ and a subsequent shift by $\vec{q}$. Thus, a point $\vec{p}|_p$ is

$$\vec{p}|_m = \begin{bmatrix} \cos\rho & \sin\rho \\ -\sin\rho & \cos\rho \end{bmatrix} \vec{p}|_p - \vec{q} \tag{2.10}$$

in the conic center frame. ρ and $\vec{q}$ are also defined in Section 2.5.4.

2.4.5 Angles

An intuitive way to describe the orientation of the camera with respect to the Earth are angles. In contrast to conventional HSs, this sensor system does not require any specific orientation in order to function. Therefore, it can be mounted in any arbitrary orientation with respect to the body coordinate system of the vehicle. Due to that, the conventionally used angles, pitch and roll, are replaced in favour of a more fitting set: azimuth ψ and elevation ϵ. Figure 2.3 illustrates the definitions of them in an example.

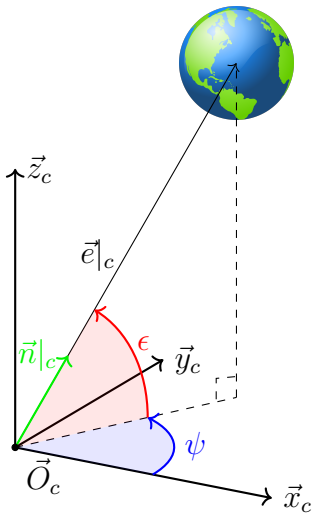


Figure 2.3: Definitions of the Azimuth ψ and Elevation ϵ Angles

If the position vector of the Earth in the camera coordinate system is $\vec{e}\,|_c$ then its

unit (nadir) vector is

$$\vec{n}|_c = \frac{\vec{e}\,|_c}{\|\vec{e}\,|_c\|} \tag{2.11}$$

and with that azimuth

$$\psi = \arctan2\left(\frac{n_y|_c}{n_x|_c}\right) \tag{2.12}$$

is defined as the angular deviation of the nadir vector in the $\vec{x}_c\vec{y}_c$-plane from the $\vec{x}_c$ axis and elevation

$$\epsilon = \arcsin(n_z|_c) \tag{2.13}$$

as the angle between the nadir vector and its projection into the $\vec{x}_c\vec{y}_c$-plane. The range of azimuth is therefore $\psi \in [0, 2\pi[$ and that of the elevation $\epsilon \in [-\pi/2, \pi/2]$. For a full attitude information the angle of rotation around the nadir vector is necessary but with the method presented in this thesis that angle is not observable.

2.5 Geometrical Considerations

To understand the connection between the projection of the horizon and the attitude of the camera some theoretical considerations are necessary. In the following, the entire concept is presented in detail. For a clearer view, all items in this section are defined in the camera coordinate system if not noted otherwise.

2.5.1 Focal Length

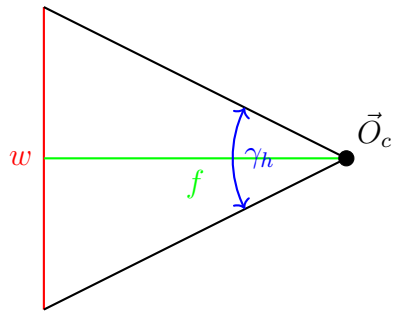


Figure 2.4: Relation of Horizontal FOV and Focal Length

The field of view (FOV) and the focal length f can be translated into each other if the image dimensions are known. Figure 2.4 illustrates the geometrical connection of the items. With that knowledge, one can formulate the expression for the focal

length

$$f = \frac{w}{2\tan\left(\frac{\gamma_h}{2}\right)} = \frac{h}{2\tan\left(\frac{\gamma_v}{2}\right)} \tag{2.14}$$

where γ_h and γ_v is the horizontal and vertical FOV respectively. By this expression the unit of the focal length is given in pixels.

2.5.2 Definition of Cone

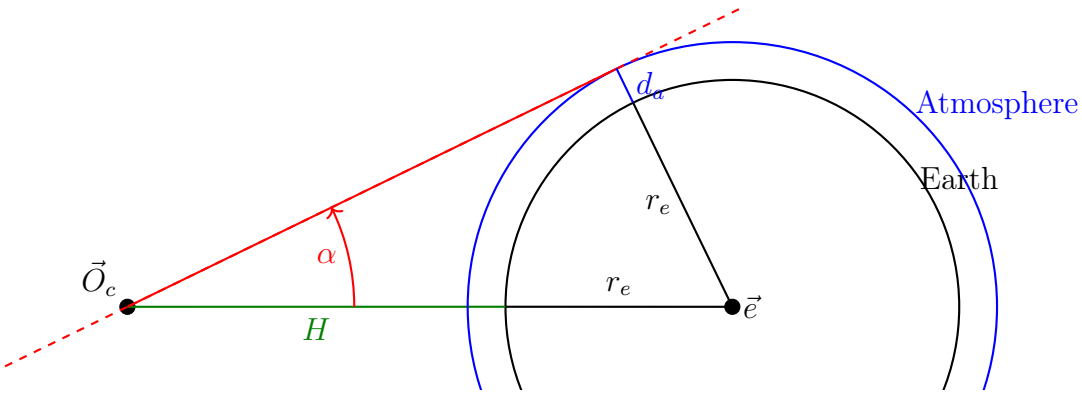


Figure 2.5: Definition of the Cone

The pinhole camera model states, there is a focal point that must be intersected by all light rays that construct the image. A point on the surface of the sphere is visible, if it can be connected with the focal point in a straight line which does not intersect the sphere. The set of all visible points forms a spherical cap and the set of all lines forms a solid cone. Lets consider only the surface of that cone and extend it infinitely in both directions. This infinite cone is the set of all tangents to the sphere and through the focal point. In this approach it is assumed that the Earth is a sphere with the position vector $\vec{e}$ in the camera coordinate system and the radius r_e. The origin of the camera coordinate system is in the focal point $\vec{O}_c$ of the optical system. That means, all light rays have to go through the origin. An infinite cone can be defined with a center point, a vector and an angle. Figure 2.5 illustrates the geometry for simplicity in 2D but it can easily be extended to 3D if one imagines rotating the sketch around the $\vec{e}$ vector. The cone can be mathematically described by

$$(\vec{e} \cdot \vec{p})^2 - \vec{e}^{\,2}\vec{p}^{\,2}\cos^2\alpha = 0 \tag{2.15}$$

where

$$\alpha = \arcsin\left(\frac{r_e + d_a}{r_e + H}\right) \tag{2.16}$$

is the semi-opening angle of the cone as indicated in Figure 2.5, $\vec{p}$ is a point on the surface of the cone and d_a is the thickness of the atmosphere. It is important to include the atmosphere in the model, because the horizon detection actually acquires a brightness gradient in the atmosphere instead of the Earth's surface. With the substitution

$$\vec{n} := \frac{\vec{e}}{\|\vec{e}\|} \tag{2.17}$$

this definition can be reformulated as

$$\|\vec{e}\|^2 (\vec{n} \cdot \vec{p})^2 - \|\vec{e}\|^2 \vec{n}^2 \vec{p}^{\,2} \cos^2\left(\arcsin\left(\frac{r_e + d_a}{r_e + H}\right)\right) = 0 \tag{2.18}$$

where $\vec{n}$ is the line-of-sight (nadir) vector in the camera coordinate frame. For another substitution

$$\chi := 1 - \left(\frac{r_e + d_a}{r_e + H}\right)^2 \tag{2.19}$$

the equation results in the simple form

$$(\vec{n} \cdot \vec{p})^2 = \chi \vec{p}^{\,2} \tag{2.20}$$

which has the advantage that the direction and the length of the vector from the camera origin to the center of the Earth are separated.

2.5.3 Parametrization of the Projected Curve

The projection of the horizon in the image plane is the intersection of the cone's surface with the image plane $p_z = f$. The curve of the intersection is described by the conic section

$$Ap_x^2 + 2Bp_xp_y + Cp_y^2 + 2Dp_x + 2Ep_y + F = 0 \tag{2.21}$$

with the parameters

$$A = n_x^2 - \chi \tag{2.22}$$

$$B = n_x n_y \tag{2.23}$$

$$C = n_y^2 - \chi \tag{2.24}$$

$$D = n_x n_z f \tag{2.25}$$

$$E = n_y n_z f \tag{2.26}$$

$$F = f^2(n_z^2 - \chi). \tag{2.27}$$

This equation is the mathematical description of the projected curve of the Earth's horizon onto an image if the optical system with a focal length f is at an altitude H and the Earth center is in $\vec{n}$ direction.

2.5.4 Transformation of the Principal Axis

The mixed term of Equation 2.21 can be eliminated by rotating the projection with a neatly chosen angle ρ. It can be shown that the angle must satisfy

$$\tan 2\rho = \frac{2B}{A - C}. \tag{2.28}$$

In this specific application this is true if

$$\tan \rho = \frac{n_y}{n_x}. \tag{2.29}$$

Since $p_z = f$ the coordinate in the principle point coordinate system is just $\vec{p}|_p = \begin{pmatrix} p_x & p_y \end{pmatrix}^T$. Applying the rotation

$$\vec{p}|_r = \begin{bmatrix} \cos\rho & -\sin\rho \\ \sin\rho & \cos\rho \end{bmatrix} \vec{p}|_p \tag{2.30}$$

gives a new equation of the conic section in the form

$$A_1 \left(p_x|_r\right)^2 + C_1 \left(p_y|_r\right)^2 + 2D_1 p_x|_r + 2E_1 p_y|_r + F = 0 \tag{2.31}$$

with the coefficients

$$A_1 = \vec{n}_x^2 + \vec{n}_y^2 - \chi \tag{2.32}$$

$$C_1 = -\chi \tag{2.33}$$

$$D_1 = \vec{n}_z f \sqrt{\vec{n}_y^2 + \vec{n}_x^2} \tag{2.34}$$

$$E_1 = 0. \tag{2.35}$$

Applying a subsequent coordinate shift of

$$\vec{p}|_r = \vec{p}|_m + \vec{q} \tag{2.36}$$

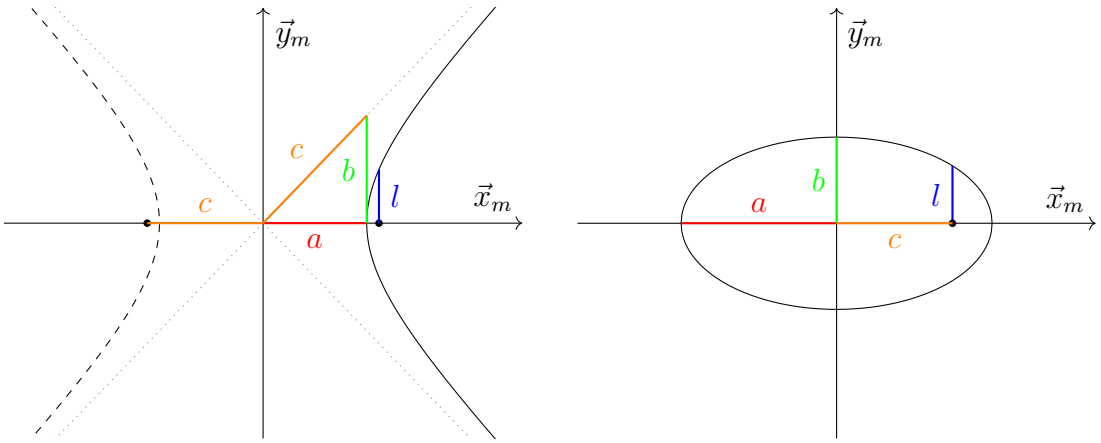
where $\vec{q} = \begin{pmatrix} -\frac{D_1}{A_1} & 0 \end{pmatrix}^T$ eliminates the remaining linear term and reveals the conic section in the form

$$A_1 \left(p_x|_m\right)^2 + C_1 \left(p_y|_m\right)^2 + F_1 = 0 \tag{2.37}$$

where

$$F_1 = \frac{f^2(1-\chi)\chi}{n_x^2 + n_y^2 - \chi}. \tag{2.38}$$

2.5.5 Classification of the Conic Section



(a) Hyperbola (b) Ellipse

Figure 2.6: Hyperbola & Ellipse

Equation 2.37 is a parametric form of the conic section in the conic centre coordinate system. In this form it is much easier to classify the type of conic section. Figure 2.6

illustrates an example hyperbola and ellipse for better understanding. Lets assume that the camera cannot be in or below the earth surface ($H > 0$) and let $d_a = 0$. This implies that

$$C_1 < 0, \quad F_1 \neq 0 \quad \text{and } \operatorname{sign} A_1 \neq \operatorname{sign} F_1. \tag{2.39}$$

To classify the conic section lets first regard the case **(I)** ($\mathbf{A_1 < 0}$)**:**
Equation 2.37 can be written as

$$\frac{(p_x|_m)^2}{\left(\sqrt{\frac{F_1}{|A_1|}}\right)^2} + \frac{(p_y|_m)^2}{\left(\sqrt{\frac{F_1}{|C_1|}}\right)^2} = 1 \tag{2.40}$$

and represents an **ellipse** with rotation ρ, center point

$$\vec{O}_m|_p = \frac{-fn_z}{n_x^2 + n_y^2 - \chi} \begin{pmatrix} n_x \\ n_y \end{pmatrix}, \tag{2.41}$$

semi-major-axis

$$a = \sqrt{\frac{F_1}{|A_1|}} = \frac{f\sqrt{(1-\chi)\chi}}{n_x^2 + n_y^2 - \chi} \tag{2.42}$$

and semi-minor-axis

$$b = \sqrt{\frac{F_1}{C_1}} = f\sqrt{1 + \frac{n_z}{n_x^2 + n_y^2 - \chi}}. \tag{2.43}$$

In the case **(II)** ($\mathbf{A_1 > 0}$) the equation can be written as

$$\frac{(p_x|_m)^2}{\left(\sqrt{\frac{F_1}{|A_1|}}\right)^2} - \frac{(p_y|_m)^2}{\left(\sqrt{\frac{F_1}{|C_1|}}\right)^2} = 1 \tag{2.44}$$

and represents a **hyperbola** with rotation ρ, center point $\vec{O}_m$ and semi-axes a and b as derived in **(I)** above.
In the last case **(III)** ($\mathbf{A_1 = 0}$) the coordinate shift in Equation 2.36 is not possible but the equation of the conic section in Equation 2.31 becomes

$$C_1 (p_y|_r)^2 + 2D_1 p_x|_r + F = 0. \tag{2.45}$$

With a coordinate shift

$$\vec{p}|_r = \vec{p}|_m + \vec{q}_1 \tag{2.46}$$

where $\vec{q}_1 = \begin{pmatrix} -\frac{F}{2D_1} & 0 \end{pmatrix}^T$ the equation is reduced to

$$(p_y|_m)^2 = -2\frac{D_1}{C_1} p_x|_m \tag{2.47}$$

and represents a **parabola**.

In one sentence: If the Earth is intersected by the $\vec{x}_c\vec{y}_c$-plane (image plane), the projection is a hyperbola, if not, it is an ellipse and if the earth center is in the $\vec{x}_c\vec{y}_c$-plane, the result is a parabola. Due to the limited numeric precision of machines the case that produces a parabola is very unlikely.

2.5.6 Hyperbola Branch

Obviously, the hyperbola has two branches but only one of them actually corresponds to the projection of the horizon. Solving Equation 2.37 for $p_x|_m$ yields

$$p_x|_m = \pm\sqrt{(-C_1 \left(p_y|_m\right)^2 - F_1)/A_1}. \tag{2.48}$$

In this form the two branches are distinguishable by the sign. From geometrical considerations it is apparent that the equation with the positive sign is the sought curve.

2.5.7 Semi-Latus-Rectum

For an illustration of the following items please refer to Figure 2.6. The linear eccentricity c is a measure of flatness of hyperbolae and ellipses. For hyperbolae it is defined as

$$c^2 = a^2 + b^2 \tag{2.49}$$

and for ellipses defined as

$$c^2 = a^2 - b^2. \tag{2.50}$$

The two focal points (foci) of a conic section are points on the $\vec{x}_m$-axis with distance c from origin. The distance of the foci to the curve in $\vec{y}_m$-direction is called the semi-latus-rectum l and is defined as

$$l = \frac{b^2}{a}. \tag{2.51}$$

With Equation 2.42 and Equation 2.43 the Semi-Latus-Rectum can be written as

$$l = \frac{f(d_a + r_e)}{\sqrt{(d_a + r_e)^2 - (H + r_e)}}. \tag{2.52}$$

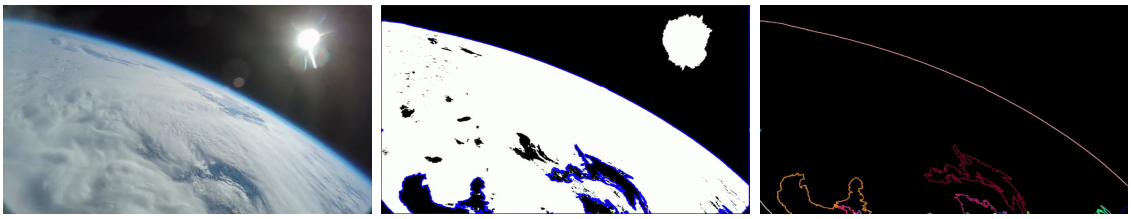
2.6 Edge Detection (and Threshold Filter)

The edge detection is a topological search based on the algorithm introduced by Suzuki and Abe [1983] and adapted for this application in [Barf, 2014]. It searches for pixels that are at the edge between a white and a black area in a binary image. In the version of the algorithm introduced in this thesis work, the threshold filter step is performed by the edge detection. This has the advantage, that only those pixels are converted to binary that are actually accessed during the edge detection process. This idea was initially proposed by Wagner et al. [2016]. Due to the way this search operates the fraction of pixels that are converted is very small. A threshold filter is applied to a colored pixel by comparing the mean of all color values to a threshold value μ_t. Higher values cause a 1 in the binary image and lower values a 0.

The general functional principle of the topological search is to iterate through pixels and search for a change from zero to one or vice-versa. Having found such a pixel, the border is followed and the pixels marked until the starting pixel is reached. After that, the search is continued. The original algorithm starts its search in the left upper corner and iterates through all pixels to search for an edge pixel. This however is a much smaller search grid than necessary since the sought projection is large.

Therefore, before starting the search algorithm, the smallest possible diameter, the minor-axis $2b$, of the projection is determined. The semi-minor-axis b is approximated with the semi-latus-rectum l defined in Section 2.5.7. The clue about this item is, that it is independent of the actual attitude. Only constant values and the altitude above ground H are required which is assumed to be available a priori knowledge. The assumption $l \approx b$ is sufficiently accurate in this application, for the following reasons. Since $l = \frac{b^2}{a} \leq b$ is by definition always true, the semi-latus-rectum is an underestimation of the semi-minor-axis. Thus a grid with size $2l$ will always intersect the projection. The algorithm searches all image borders and additionally all horizontal rows that are at distance $k \cdot 2l$, $k \in \{1, 2, 3, ..., h/2l\}$ from the upper border. Of course, these horizontal lines are only required at high altitudes where it is possible that the projection is completely inside the image and does not touch any image border. In those cases the projection must be a nearly circular ellipse where the above approximation is very accurate ($l \approx b \approx a$). In the case of hyperbolae or parabolae the horizontal lines are not required because they are infinite and will always intersect the border. Since the observed hyperbolae are very flat, the latus-rectum is larger than the image height and thus no unnecessary horizontal searches are performed. The altitudes in which these extra search lines are required are illustrated in Figure 4.3 as the red hatched area. Outside that area the grid search is reduced to the image borders. Not only does this optimization save computational time but it also reduces clutter since smaller objects in the image are less likely to be found.

Another feature of this algorithm, as described in [Suzuki and Abe, 1983] is that inner edges are ignored. Since pixels that have been found during the border following process are marked, the grid search can ignore all found edge pixels between two markers. This similarly reduces computational time and clutter without compromising reliability since the horizon line can never be an inner border. In Figure 2.9 the working principle is illustrated in an example. Figure 2.7b is the binary image



(a) Original (b) Binary (c) Edges

Figure 2.7: Example of the Working Principle of the Edge Detection

of Figure 2.7a if one would apply the threshold filter to all pixels. The pixels that are marked blue are those that have been accessed by the edge detection. These are those at the image rim and the edge pixels of outer borders found in the grid search. As discussed above, the number of blue pixels is much smaller compared to the total amount of pixels. Moreover, the white area produced by the sun was not found because it is smaller than the grid size and does not touch any border. All of the inner borders of the black spots inside the Earth area have been ignored. The spots at the lower border of the image that are marked blue are no inner borders. Since their black area is connected to the lower image border, the border following process detects them as outer border of the big white area.
Figure 2.7c illustrates another feature of the algorithm: the distinction of lines. All pixels in a line found by the border following process that are at the image border ($x = 0 \vee x = w-1 \vee y = 0 \vee y = h-1$) are removed and the remaining line fragments are regarded as individual lines. The individual lines are indicated by different colors in Figure 2.7c.
For every sub-line of every line found by the border following process the best fitting nadir vector is calculated (cf. Sec. 2.9). But before that, all points are undistorted (cf. Sec. 2.7) and all outliers are removed (cf. Sec. 2.8). The vector with the smallest residual (cf. Sec. 2.10) is selected as result if the plausibility check (cf. Sec. 2.11) is approves.

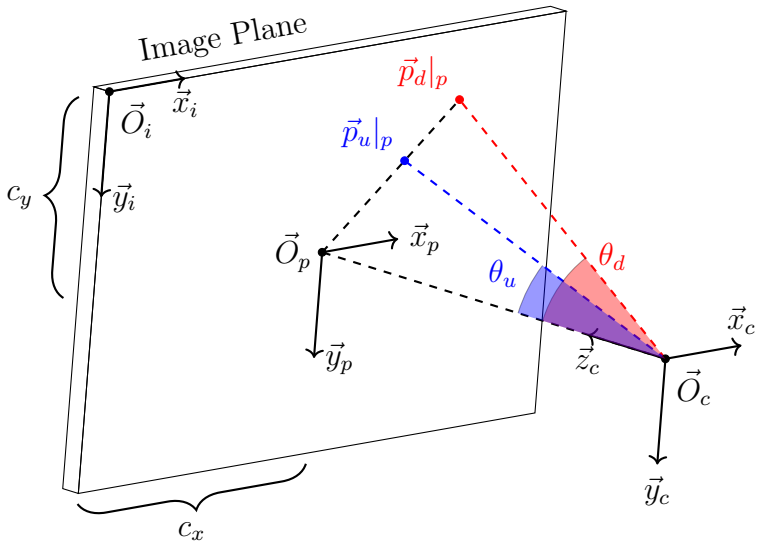


Figure 2.8: Undistortion of point

2.7 Undistortion

Any real optical system introduces distortions to the image that can be modelled with the pinhole-camera model. However, the fisheye lens poses here a speciality because it introduces much higher distortions then standard lenses. The commonly used distortion model by Zhang [2000] is therefore not applicable and is replaced by the more fitting generic distortion model by Kannala and Brandt [2006]. This distortion model was also chosen because there exists a calibration routine within the OpenCV library. Kannala and Brandt introduced a radially symmetric distortion model where the distortion depends only on the angle θ_u between principal axis and the incoming ray as illustrated in Figure 2.8. Please note that in the following derivation the subscript u and d stand for undistorted and distorted respectively. An undistorted perspective projection is given by

$$\|\vec{p}_u|_p\| = f \tan\theta_u \tag{2.53}$$

where f is the focal length. The model by Kannala and Brandt takes a very generic approach of the general form

$$\theta_d = k_0\theta_u + k_1\theta_u^3 + k_2\theta_u^5 + k_3\theta_u^7 + k_4\theta_u^9 \ldots \tag{2.54}$$

where the coefficients k_i are called distortion coefficients. In the implementation of the calibration routine of the OpenCV library k_0 is set to $k_0 = 1$ and only the coefficients k_1,k_2,k_3 and k_4 are estimated using the Levenberg-Marquardt algorithm (LMA). Along with the distortion parameters the algorithm also estimates the camera

matrix

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.55}$$

where f_x and f_y are the estimated focal lengths in the horizontal and vertical direction respectively and the point (c_x, c_y) is the estimated image center. The camera matrix is basically the transformation matrix to transform a point from the camera coordinate system into the image coordinate system while taking the calibrated intrinsic parameters into account. The transformation

$$\vec{p}|_i = K \frac{\vec{p}_d|_c}{p_{d,z}|_c} \tag{2.56}$$

is similar to the one derived in Section 2.4.2. Since the transformation is not bidirectional, the transformation in the opposite direction

$$\vec{p}_d|_p = f \begin{pmatrix} (p_{d,x}|_i - c_x)/f_x \\ (p_{d,y}|_i - c_y)/f_y \end{pmatrix} \tag{2.57}$$

results in a point in the principal point coordinate system.
The LMA requires multiple sets of reference points which are substitute to the distorted projection and whose real relative distance to each other is known. These reference points are commonly generated by taking photos of a well known chessboard patterns. The corners of the squares can easily be detected in the image and their relative distance is known.
Of course the undistortion for the whole image is way too computational intensive for an embedded system that is required to perform in near real-time. Fortunately, it is not necessary to undistort every single pixel in the image. The edge detection itself can be performed on the distorted image and only the subset of pixels that are identified as edges must be undistorted. In this radial distortion model the distortion of a point can be seen as a scaling with factor

$$s = \frac{\|\vec{p}_u|_p\|}{\|\vec{p}_d|_p\|} = \frac{\tan\theta_u}{\hat{p}} \tag{2.58}$$

along the line to the principal point where

$$\hat{p} = \frac{\|\vec{p}_d|_p\|}{f}. \tag{2.59}$$

Unfortunately, there is no simple closed-form analytical solution for θ_u, therefore it is approximated by the iterative numeric Algorithm 2.1. The number of iteration $\mu_{Ui} = 10$ was found to be reasonable. The undistorted point in the principal point coordinate system is then given by

$$\vec{p}_u|_p = s \cdot \vec{p}_d|_p. \tag{2.60}$$

Algorithm 2.1: Algorithm to determine θ_u

function $\theta_u(\hat{p},\mu_{Ui},k_1,k_2,k_3,k_4)$
  $\theta_u = \hat{p}$;
  foreach $i \in [1, \mu_{Ui}]$ **do**
    $\theta_u = \hat{p}/(1 + k_1\theta_u^2 + k_2\theta_u^4 + k_3\theta_u^6 + k_4\theta_u^8)$;
  end
  return θ_u;

2.8 Outlier Detection

(a) Undistorted Lines (b) RANSAC Line (c) Result

Figure 2.9: Example of the Working Principle of the Outlier Detection

When the sun is in the picture it creates major disturbances as can be seen in Figure 2.9a. It can not always be assumed that the best threshold is found and the horizon is strictly separated from the disturbances. This is why an algorithm is needed to make the detection less dependent on the threshold value.
The RANSAC algorithm by Fischler and Bolles is "a paradigm for fitting a model to experimental data" with "a significant percentage of gross errors" [Fischler and Bolles, 1981]. The algorithm adapted to this specific application works like this:

1. randomly select three pixels in the horizon line
2. estimate the nadir vector from those three pixels
3. calculate the conic section this vector corresponds to
4. count all pixels that are closer to the conic than a certain threshold μ_{Rcd}
5. repeat steps 1-4 μ_{Ri}-times
6. select the vector that corresponds to the most pixels and estimate the vector with all pixels in its corresponding conic's proximity
7. return this new vector

The threshold μ_{Rcd} should be only a few pixels and the iteration number μ_{Ri} can be calculated with

$$\mu_{Ri} = \frac{\log(1 - \mu_{Rdp})}{\log\left(1 - (1 - \mu_{Ror})^3\right)} \tag{2.61}$$

where $\mu_{Rdp} \in]0, 1[$ is the probability that the correct part of the horizon line is selected and $\mu_{Ror} \in]0, 1[$ is the ratio of outliers in the horizon line [Fischler and Bolles, 1981]. E.g. for a success probability of 80% and an outlier ratio of 30% the number of iterations is only 4 but for a success probability of 99% and an outlier ratio of 70% the number of iterations must already be as high as 169.
With this algorithm it is possible to find the horizon even if there are heavy disturbances at the horizon line as illustrated in Figure 2.9. The left image shows all undistorted lines found in an example images. The long red line is heavily cluttered by the sun and its reflections in the protective glass in front of the camera. The image in the center shows those parts of the line that were identified to be part of the horizon by the RANSAC algorithm. As can be seen, the algorithm perfectly removed all clutter and left only the horizon line.

2.9 Vector Estimation

In this section all items are defined in the camera coordinate system if not noted otherwise. Equation 2.20 in matrix notation is

$$\vec{n}^{\,T} \left(\vec{p}_h \vec{p}_h^{\,T}\right) \vec{n} = \chi \vec{p}_h^{\,T} \vec{p}_h \tag{2.62}$$

where $\vec{p}_h = \begin{pmatrix} \vec{p}_h|_p & f \end{pmatrix}^T$ if $\vec{p}_h|_p$ is the coordinate of a pixel in the horizon line P_H. This over-determined system of equations can be solved with a non-constraint, iterative, non-linear least-squares fitting algorithm. The k-th iteration of the algorithm returns the best fitting vector $\vec{e}_{k+1}|_c$ which is then used as start solution for the $(k+1)$-th iteration. The least-squares ansatz can be formulated as

$$\Delta\vec{e}_k = \left(H_k^T H_k\right)^{-1} H_k^T \Delta\vec{y}_k \tag{2.63}$$

where Δe_k is the deviation of the vector $\vec{e}_k$ to the best fitting one and

$$\Delta\vec{y}_k = \chi \begin{pmatrix} \vec{v}_1^{\,T} \vec{v}_1 \\ \vdots \\ \vec{v}_{|P_H|}^{\,T} \vec{v}_{|P_H|} \end{pmatrix} - \begin{pmatrix} \vec{e}_k^T M_1 \vec{e}_k \\ \vdots \\ \vec{e}_k^T M_{|P_H|} \vec{e}_k \end{pmatrix} \tag{2.64}$$

where the matrix $M_i = \vec{v}_i \vec{v}_i^{\,T}$ must be calculated for every data point $i \in [1, |P_H|]$. The matrix H_k is given by

$$H_k = \begin{bmatrix} 2\vec{e}_k^{\,T} M_1 \\ \vdots \\ 2\vec{e}_k^{\,T} M_{|P_H|} \end{bmatrix}. \tag{2.65}$$

The result for each iteration is calculated by

$$\vec{e}_{k+1} = \vec{e}_k + \Delta\vec{e}_k \tag{2.66}$$

which is then used as starting solution for the next iteration. As starting solution for the very first iteration the vector $\vec{e}_1 = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$ is used because it points in approximately a similar direction as the sought vector. The algorithm is terminated if $\Delta\vec{e}_k$ drops below a certain threshold μ_{Ec} or exceeds a limit μ_{Ei} of iterations.
Theoretically, the altitude H could be estimated with this method as well. But since this figure is assumed to be known very accurately it is rather used to give a better start solution and to guide the fitting process. Although the constant χ which contains H is part of the fitting process, the fit is not forced to a solution that has the altitude H. The length $\|\vec{e}_k\|$ of the fitted vector reflects the deviation of the measured altitude from the expected one. This way the estimated nadir vector $\vec{n}_e = \frac{\vec{e}_k}{\|\vec{e}_k\|}$ is independent from estimation errors of the altitude H, Earth radius r_e or thickness d_a of the visible atmosphere.

Prior to this approach there were multiple attempts to fit the parameters A,B,C,D,E and F of a general conic section to the dataset and calculate the attitude information with Equations 2.22 - 2.27. That however requires to constrain the least-squares fit to avoid the trivial solution where all parameters are zero. As described by Fitzgibbon et al. [1996] there are multiple ways to achieve that. Maybe the most simple way and primarily one that allows to fit ellipses, parabolas and hyperbolae is the constrain $F = 1$ introduced by Rosin [1993]. The fitting itself, which could be realized in a non-iterative and closed-form way, delivered satisfactory results whereas the calculation of the nadir vector was not conclusive. The main reason for that is the high amount of degrees of freedom in the fitting process. Therefore, the process was restricted more tightly in order to reduce the degrees of freedom by formulating Equations 2.22 - 2.27 as restrictions. Unfortunately, these restrictions created many local minima in the error function which act as trap for the, now required, iterative least square fit and thus made this process unusable. A formal derivation of both of these approaches can be found in Appendix A. The clue about the solution presented here is that it does not actually fit a conic section to a set of data but rather fits a nadir vector that corresponds to a conic section that fits the data. This reduces the degrees of freedom to three but requires an iterative process due to the non-linearity of the problem statement (Eqn. 2.62).

It must also be noted, that this is an algebraic fit which means that the algebraic error function is minimized. The algebraic error, further explained in Section 2.10, however is not the most accurate way to describe the distance of a point to a conic section as stated by multiple authors including Zhang. But the proposed method, geometric fitting, is very complex and was therefore not implemented in this thesis, also because the results with the algebraic fit were most satisfying. One drawback of the algebraic error is that there is no distinction between the two branches of a hyperbola as also mentioned in Section 2.10. Fortunately, with the x-distance, proposed in the same section, the outlier detection (cf. Sec. 2.8) selects points based on their distance to the correct branch and thus the final result is calculated with points only close to that branch.

2.10 Residual

The residual is a non-negative value that provides a measure of the quality of the fit. The closer the residual is to zero the better is the quality. It is defined as the sum of the squared error values (distances) δ_i of all measurements.

$$\lambda_\delta = \sum_{i=0}^{|P_H|} \delta_i^2 \tag{2.67}$$

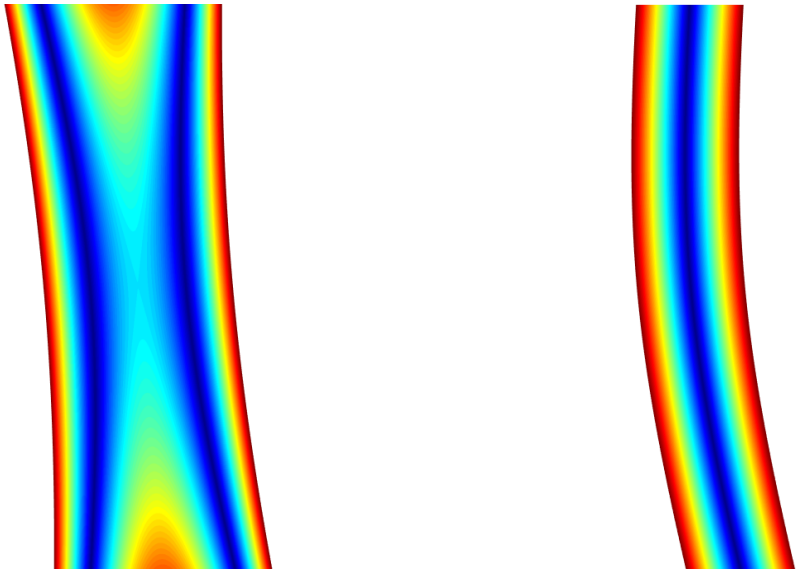
There are, however, multiple ways to interpret the error δ. In the literature the terms algebraic and geometric (or euclidian) residual are used.

2.10.1 Algebraic Distance

The algebraic distance of a measurement $\vec{p_k}|_p \in P_H$ refers to the magnitude of the error in the equation of the general conic section and is defined as

$$\delta_{b,k} = Ap_{x,k}^2 + 2Bp_{x,k}p_{y,k} + Cp_{y,k}^2 + 2Dp_{x,k} + 2Ep_{y,k} + F \tag{2.68}$$

where the parameters A-F are given in Equations 2.22 - 2.27. Those equations require the nadir vector (unit vector) and the altitude. The vector estimation in Section 2.9 however returns a vector that is not a unit vector because it also contains the error of the altitude approximation and the thickness of the atmosphere. In order to include the error of the detected altitude to the expected altitude in the residual, the parameters A-F are determined with the normalized vector provided by the vector estimation and the known altitude. Unfortunately, the algebraic distance is not a very accurate measure of distance as Figure 2.10a suggests. The gradient



(a) Algebraic Residual (b) X Residual

Figure 2.10: Qualitative Plot of the Magnitude of Residuals (red: high, blue:low)

of the magnitude of the distance is not perpendicular to the curve and it can not be differentiated between the branches. That means the magnitude of the algebraic distance of a point to the hyperbola is not only dependent on its Euclidian distance but also on its position. For example a point close to the second branch has a much lower algebraic distance than a point that has the same Euclidian distance to both branches.

2.10.2 Geometric Distance

The geometrical distance is the actual Euclidian distance between the coordinate $\vec{p}_k$ and its orthogonal projection $\vec{b}_k$ in the curve

$$\delta_{g,k} = \|\vec{p}_k - \vec{b}_k\|. \tag{2.69}$$

In Figure 2.11 the geometric distance is illustrated as dashed line. Unfortunately the determination of the orthogonal projection is a very complex process. Zhang [1996] presents a concept to determine $\vec{b}_k$ but it is very computational intensive. Therefore, although the geometrical distance is the best measure of the residual, it is not considered in this thesis.

2.10.3 X-Distance

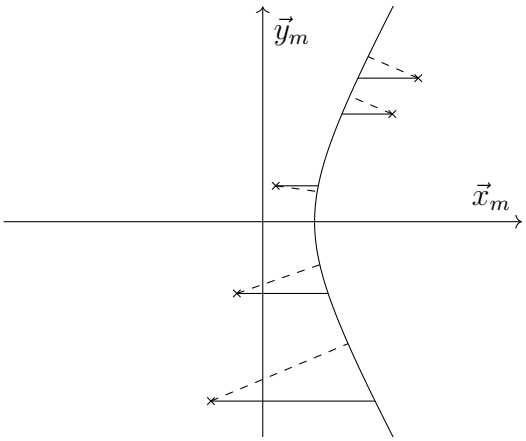


Figure 2.11: Geometrical (dashed) and X Distance

Because the algebraic residual does not work for hyperbolae and the geometric distance is computationally too expensive, a new distance is proposed: the x-distance. As illustrated in Figure 2.11 it is simply the distance of $\vec{p}_k|_m$ to the curve in $x|_m$ only. It is therefore directly derived from Equation 2.48 and is defined as

$$\delta_{a,k} = p_{x,k}|_m - \sqrt{(-C_1 \left(p_{y,k}|_m\right)^2 - F_1)/A_1}. \tag{2.70}$$

The parameters A-F are determined in the same manner as explained for the algebraic distance. This is obviously not the most accurate way to determine a distance but is accurate enough for this application for two reasons. Firstly, as Figure 2.11 suggests, the error in distance $\Delta\delta = |\delta_g - \delta_a|$ is dependent on the flatness of the hyperbola. The larger the ratio of $\frac{b}{a}$ the smaller is $\Delta\delta$. Secondly, $\Delta\delta$ is also dependent on the position of the point. The closer the point is to the center of the conic section the smaller is $\Delta\delta$. In this application both cases are true, very flat hyperbolae and measurement points close to the conic center. In Figure 2.10b it can be observed, that the gradient of the residual aligns with the perpendicular line to the curve. The x-distance is a good trade-off between accuracy and computational intensity when calculating the residual of a flat hyperbolae. For ellipses and less eccentric hyperbolae this distance is too inaccurate, therefore the algebraic distance is used in those cases.

2.11 Plausibility Check

In order to filter out erroneous detections each result has to undergo a plausibility check that determines whether the projected curve, corresponding to the calculated attitude, would be in the image. In cases where the attitude information is calculated from a line that is not the horizon this might not be the case and the result can be discarded immediately. Figure 2.12 shows some examples of how conic sections may lie in the image plane. The grey area represents the image area inside the image

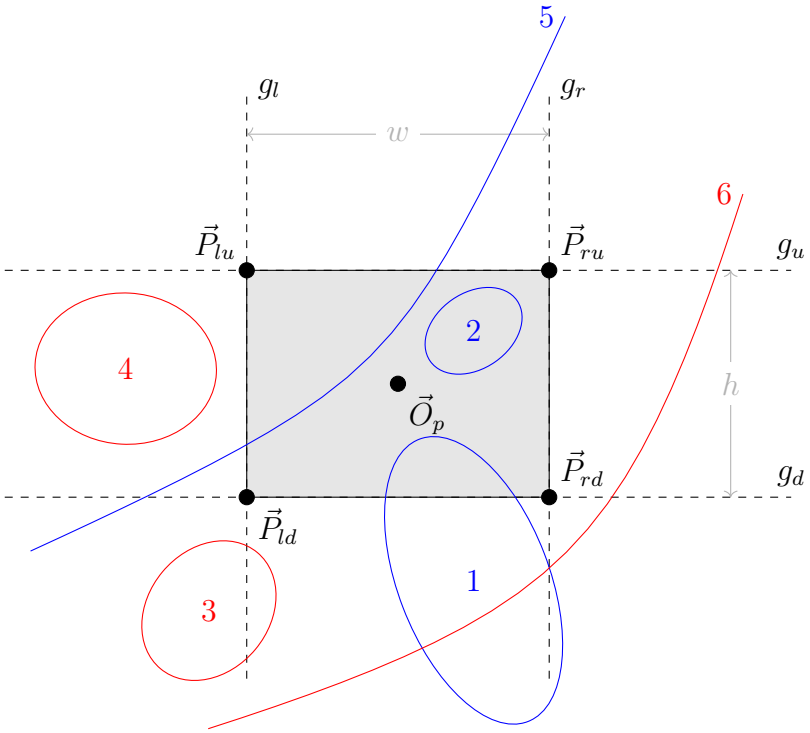


Figure 2.12: Examples of conic sections in the image plane

plane that has a width of w and a height of h. The curves that are in the image are marked blue and the ones that are not, are red. The curves with the numbers 5 and 6 are hyperbolae and 1-4 are ellipses. The image corners are the points P with indexes l or r for left and right with subsequent u or d for up and down. The verification can be broken down into a combination of the six logical literals given in Table 2.2. For the horizon to be in the image it is crucial that $\mathcal{Z}$ is true. This might seem obvious and superfluous but the mathematical concept allows objects behind the camera to be projected onto the image plane. That case is eliminated by requiring $\mathcal{Z}$ in any case. If so, there are two possibilities: either the conic section intersects the image borders ($\mathcal{G}$) or it does not. In the first case the horizon must be visible. In the second

Literal	Description
$\mathcal{Z}$	The horizon is inside the maximum field of view of the camera
$\mathcal{G}$	The conic section intersects one of the image borders
$\mathcal{Q}$	The centre of the conic section $\vec{O}_m$ is in the image
$\mathcal{K}$	The conic section intersects one of the infinite lines along the image borders.
$\mathcal{H}$	The conic section is an hyperbola
$\mathcal{V}$	The horizon is visible

Table 2.2: Literals for the Plausibility Check

case there are again two cases. First, the conic section is an ellipse and is completely inside the image borders or the conic section is completely outside the image. The first case is true if no intersection with any line exists. The statement if an horizon is in the image or not is thus:

$$\mathcal{V} := \mathcal{Z} \wedge (\mathcal{G} \vee (\mathcal{Q} \wedge \neg\mathcal{K})) \tag{2.71}$$

The literal $\mathcal{Z}$ can be determined by analysing the maximum elevation at which the earth is still in the FOV and and the minimum elevation at which the horizon leaves the FOV and only the earth surface is visible. With that consideration, $\mathcal{Z}$ can be defined as

$$\mathcal{Z} := \left|\frac{\pi}{2} - \epsilon - \alpha\right| < \frac{\gamma_h}{2}. \tag{2.72}$$

From case (II) of the classification in Section 2.5.5 it is given that

$$\mathcal{H} := A_1 < 0. \tag{2.73}$$

By looking at the sketch in Figure 2.12 it is also easy to determine that

$$\mathcal{Q} := \left(-\frac{w}{2} < O_{m,x}|_p < \frac{w}{2}\right) \wedge \left(-\frac{h}{2} < O_{m,y}|_p < \frac{h}{2}\right). \tag{2.74}$$

To derive the remaining two literals, the intersections of the conic section with the four lines

$$\vec{g}_l = \vec{P}_{lu} + \lambda_l \left(\vec{P}_{lu} - \vec{P}_{ld}\right) \tag{2.75}$$

$$\vec{g}_r = \vec{P}_{ru} + \lambda_r \left(\vec{P}_{ru} - \vec{P}_{rd}\right) \tag{2.76}$$

$$\vec{g}_u = \vec{P}_{lu} + \lambda_u \left(\vec{P}_{lu} - \vec{P}_{ru}\right) \tag{2.77}$$

$$\vec{g}_d = \vec{P}_{ld} + \lambda_d \left(\vec{P}_{ld} - \vec{P}_{rd}\right) \tag{2.78}$$

are analyzed. The intersection points I are indexed with $[l|r|u|d]$ according to the line and with a subsequent $[1|2]$ since there are always two possible intersections. The intersection points are given by

$$\vec{I}_{u1} = \begin{pmatrix} \frac{-2D+Bh-\sqrt{s_u}}{2A} & -\frac{h}{2} \end{pmatrix}^T \tag{2.79}$$

$$\vec{I}_{u2} = \begin{pmatrix} \frac{-2D+Bh+\sqrt{s_u}}{2A} & -\frac{h}{2} \end{pmatrix}^T \tag{2.80}$$

$$\vec{I}_{d1} = \begin{pmatrix} \frac{-2D-Bh-\sqrt{s_d}}{2A} & \frac{h}{2} \end{pmatrix}^T \tag{2.81}$$

$$\vec{I}_{d2} = \begin{pmatrix} \frac{-2D-Bh+\sqrt{s_d}}{2A} & \frac{h}{2} \end{pmatrix}^T \tag{2.82}$$

$$\vec{I}_{r1} = \begin{pmatrix} \frac{w}{2} & \frac{-2E-Bw-\sqrt{s_r}}{2C} \end{pmatrix}^T \tag{2.83}$$

$$\vec{I}_{r2} = \begin{pmatrix} \frac{w}{2} & \frac{-2E-Bw+\sqrt{s_r}}{2C} \end{pmatrix}^T \tag{2.84}$$

$$\vec{I}_{l1} = \begin{pmatrix} -\frac{w}{2} & \frac{-2E+Bw-\sqrt{s_l}}{2C} \end{pmatrix}^T \tag{2.85}$$

$$\vec{I}_{l2} = \begin{pmatrix} -\frac{w}{2} & \frac{-2E+Bw+\sqrt{s_l}}{2C} \end{pmatrix}^T \tag{2.86}$$

where the contents of the square roots are

$$s_u = 4D^2 - 4AF - 4BDh + 4AEh + (B^2 - AC)h^2 \tag{2.87}$$

$$s_d = 4D^2 - 4AF + 4BDh - 4AEh + (B^2 - AC)h^2 \tag{2.88}$$

$$s_r = 4E^2 - 4CF - 4CDw + 4BEw + (B^2 - AC)w^2 \tag{2.89}$$

$$s_l = 4E^2 - 4CF + 4CDw - 4BEw + (B^2 - AC)w^2. \tag{2.90}$$

The parameters A-F are defined in Section 2.5.3 With these definitions the literals $\mathcal{K}$ and $\mathcal{G}$ can be formulated. An intersection point exists if both coordinates are real numbers. That is the case if the content of the square root is greater or equal zero. $\mathcal{K}$ is true if at least one of the intersection points exists:

$$\mathcal{K} := \bigvee_{i=1}^{4} s_i \geq 0 \tag{2.91}$$

For $\mathcal{G}$ to be true the intersection must additionally occur at least at one of the image borders. Therefore, the intersection must firstly exist, secondly it must be within the boundaries of the image and lastly, if the conic section is an hyperbola, it must be

the correct branch. That sentence can formally be written as

$$\mathcal{G} := \bigvee_{j=[u|d],k=[1|2]} \left(s_j \geq 0 \wedge -\frac{w}{2} \leq I_{jk,x} < \frac{w}{2} \wedge \left(\neg\mathcal{H} \vee \mathcal{L}(\vec{I}_{jk}) \right) \right) \tag{2.92}$$

$$\vee \bigvee_{j=[r|l],k=[1|2]} \left(s_j \geq 0 \wedge -\frac{h}{2} \leq I_{jk,y} < \frac{h}{2} \wedge \left(\neg\mathcal{H} \vee \mathcal{L}(\vec{I}_{jk}) \right) \right)$$

where

$$L(\vec{p}) := \left(p_x = \sqrt{(-C_1 p_y^2 - F_1)/A_1} \right) \tag{2.93}$$

checks whether a point is located on the correct hyperbola branch according to Equation 2.48.

CHAPTER 3

Implementation

The implementation developed in [Barf, 2014] was used as basis for all further development. In this chapter the modifications, the new software, development environment and platforms are thoroughly described. The software, processing the image and determining the attitude, is here referred to as HorizonSensor and the test environment software is referred to as HorizonSensorTest.

3.1 System Environment

The implementation by Barf [2014] was developed to be able to run on a desktop computer in conjunction with the test environment or on an embedded system. Up until this thesis however, the code was never tested on an embedded system. This section describes the environment the software was developed on and the target platform (embedded system).

3.1.1 Desktop

The development process included three different desktop computers that were continuously used. A MacBook Pro with macOS 10.12.5 and the integrated development environment (IDE) Xcode 8.1, a Dell computer with Ubuntu 16.04 and Eclipse Neon as IDE and another Dell computer with Ubuntu 14.04 and Eclipse Kepler. On all of these computers the library OpenCV 3.1 was installed. On macOS the compiler clang 8.0.0, on Ubuntu 16.04 the compiler gcc 5.4.0 and on Ubuntu 14.04 the compiler gcc 4.8.1 were used to compile executables for each platform respectively. In order

to produce binaries for the embedded system, the GNU toolchain for BlackFin was integrated in Eclipse Kepler on Ubuntu 14.04.

3.1.2 Embedded System

The embedded system is a multi function card (MFC) that has been developed prior to this thesis by the MORABA, and is used as the brain of the sounding rocket's service system.

Purpose

The MFC's main purpose within the rocket is to handle the communication between experiment and experimenters. In addition, it is in charge of providing housekeeping data of the rocket to the operators. It was also designed to perform algorithms for attitude or navigation solutions [Wittkamp and Zigiotto, 2009]. To date, the MFC is used for communication between ground and the onboard system only, thus this thesis work is the first application that makes extensive use of the card as digital signal processor (DSP).

Hardware Layout

The main components of the euro sized printed circuit board (PCB) are a dual core Blackfin 561 DSP from Analog Devices as CPU and a large Cyclone 3 FPGA made by Altera [Wittkamp and Zigiotto, 2009]. A photo of the PCB can be seen in Figure 3.1a. The MFC provides the interfaces RS422 with up to 35 MBit/s, CAN, SPI, general purpose digital I/Os and a 10/100 MBit/s ethernet interface. A 64 MB SDRAM is accessible by both, the FPGA and the CPU. Moreover, the FPGA can act as watchdog for the CPU. The debugging interface is the extension board shown in Figure 3.1b. It provides serial communication ports, an ethernet interface to the DSP and an interface to the DSP for the Blackfin JTAG debugging chain [Wittkamp and Zigiotto, 2009].

(a) MFC PCB[Wittkamp and Zigiotto, 2009]

(b) MFC with Debugging Extension Board

Figure 3.1: Working Setup of the Embedded System

Operating System

The DSP runs the realtime operating system RODOS. It serves as simple and fast hardware abstraction layer, that can be flexibly and easily modified to meet every experiment's demands. User-created threads can communicate asynchronously with each other via the middleware. Due to its minimal design it consumes only a small fraction of CPU time [Montenegro and Dannemann, 2009].

HorizonSensor Software Integration

The HorizonSensor software described in Section 3.2 is integrated as RODOS thread in the preexisting software system. This thread initializes the `HorizonSensor` and the TCP/IP socket, handles the communication with the camera simulator via the ethernet connection and executes the horizon detection. The thread acts as host and waits for the client, the camera simulator, to connect. As soon as it connects the thread sends a request to transmit an image. It is able to receive four different packages: the `ImagePackage`, `SetParamPackage`, `GetParamCommand` and `TerminateConnectionCommand`. All packages are explained in detail in Section 3.4.1.

If a `TerminateConnectionCommand` is received, the TCP connection to the client is safely terminated and the socket is prepared for a new connection. In the case of a `SetParamPackage` or a `GetParamCommand` the received parameters are configured in the `HorizonSensor` or the current parameter set is send to the camera simulator respectively. If an `ImagePackage` is received, the containing image data is used to create an instance of the class `GrayImage`. `GrayImage` is a class that inherits from `ImageFrame` (cf. Sec. 3.2) and implements the interface to the image data. This instance of the `GrayImage` class is forwarded to the `HorizonSensor` which performs the horizon detection and attitude determination. As soon as that process is completed, the results are packed into a `ResultPackage`, send to the camera simulator and a new image is requested.

3.2 HorizonSensor Software

The HorizonSensor is an implementation in `C++` of the algorithm described in Chapter 2. It requires only standard `C++` libraries and forgoes dynamic memory allocation. The implementation is based on the work in [Barf, 2014] but was largely modified, the basic structure however stayed the same.

3.2.1 Software Design

The core component of the HorizonSensor is the `HorizonSensor` class. It contains all functions and the necessary data structures to perform the horizon detection and attitude determination. It also holds a public instance of the class `Parameter` which itself contains all parameters of the algorithm. The data and functions associated with the image is collected in the abstract class `ImageFrame`. This class is not instantiable and serves as interface between the image data and the HorizonSensor. The usage is explained in detail in Section 3.2.2. The class `ConicSection` represents a conic section and provides functions to determine the distance of a point to it, to determine if the conic is visible in an image and to transform a point between principal point coordinate system and conic center coordinate system. The class `Result` contains, as its name implies, the results of the horizon detection and attitude determination process.

Listing 3.1: Example of the Image Data Interface

```
class MyImage{

const int channels = 3;
int width;
int height;
char * data;

MyImage(): public ImageFrame(){}
MyImage(char * data, int width, int height, float altitude):
        public ImageFrame(width, height, altitude):
        width(width), height(height), data(data){}

unsigned short getGreyScalePixel(I_PointI point){

   if(point.x < 0 || point.x >= width
   || point.y < 0 || point.y >= height){
           return 0;
        }
        unsigned short output = 0;
        for(int c=0; c < channels; c++){
           output += data[point.y*width*channels + point.x*
              channels + c];
        }
   return round( output / 3.0 );
   }
};
```

3.2.2 User Interface

The user interface of the HorizonSensor slightly changed compared to the version in [Barf, 2014]. During the development of PATHOS it became clear that a more generic interface to the image data is necessary. Therefore, the class `ImageFrame` is now an abstract class with a pure virtual function that is to be implemented by an inheriting class. In this function the user must implement how the image data is to be interpreted. A typical example of an implementation of such an interface is shown in Listing 3.1. The class inherits from `ImageFrame` and implements the method `getGreyScalePixel`. That function returns the content of the pixel at coordinate `point`. Here the image has three channels (eg. RGB) and the grayscale value is determined by averaging all channels. The horizon detection and attitude determination is then started by calling the member function `findHorizon` of the `HorizonSensor` class with an instance of this class as argument (cf. Lst. 3.2).

The return value of that function is an instance of the class `Result`. It contains the

Listing 3.2: Example of the HorizonSensor User Interface

```
MyImage = image(data, 512, 512, altitude);
HorizonSensor sensor;
sensor.param.setFocalLength(focalLength);
Result myResult = sensor.findHorizon(image);
```

attitude information and more information about the number of found lines or the number of points in the horizon line. All angles in the entire code are given in radians and all length specifications are given in meters. In Section 2.4 there are multiple coordinate systems defined that are used within this algorithm. To emphasize the coordinate system a point is defined in, there is a `typedef` for every coordinate system. The nomenclature is `<system>_Point<type>` where `<system>` means the subscript abbreviation (`C`,`M`,`P`,`I`) of the coordinate system and `<type>` distinguishes between `I` (integer) and `F` floating point values. A point in the principal point coordinate system with floating point values for example is written as `P_PointF`.

In order to compile the HorizonSensor, there are five preprocessor macros that have to be set: There is the macro `HS_MAX_IMAGE_WIDTH=<value>` where `<value>` is the maximum width in pixels of the image that is to be processed and there is the macro `HS_MAX_IMAGE_HEIGHT=<value>` where `<value>` is the maximum height in pixels of the image that is to be processed. With the macro `HS_MAX_HORIZON_POINTS=<value>` the maximum number of pixels a horizon line may contain is given in `<value>`. These three macros are necessary to provide a flexible design while forgoing dynamic memory allocation. For two other macros, the user has the choice between two options each. The first one has the options `HS_DOUBLE_PRECISION_FLOATINGPOINT` and `HS_SINGLE_PRECISION_FLOATINGPOINT`. All floating point variables have been defined with the type `FloatingPoint` instead of `double` or `float`. With the just described macro all such variables can be either defined as `double` (`DOUBLE_PRECISION`) or `float` (`SINGLE_PRECISION`) variables. The `DOUBLE` option may be more accurate but slower whereas the `SINGLE` option may be less accurate but faster. The second macro has the options `HS_DEBUG` and `HS_RELEASE`. If the macro `HS_DEBUG` is defined, the HorizonSensor collects more debugging information and provides functions that are not possible on an embedded system. This macro must be defined if the HorizonSensorTest environment is used. If `HS_RELEASE` is defined the HorizonSensor can be compiled for an embedded system but does not provide certain debugging information.

Since this software is supposed to run in a realtime environment, it must be ensured that the algorithm finishes by a certain predefined time. With the current design of the algorithm this can in general not be assured. Therefore, a kill switch was added to the implementation that can be activated from outside the HorizonSensor. This kill switch is a simple pointer to a boolean variable that can be set to `false` by any

instance outside the HorizonSensor. In every major loop inside the HorizonSensor, this variable is checked and if set to false the loop is terminated. This routine was placed such that no matter when this switch is activated during the process the result is always a valid attitude information although with reduced quality. The pointer to the boolean can be forwarded to the `HorizonSensor` class as optional argument of the constructor.

The documentation of the `C++` code is available in the form of doxygen-style comments in the header files. In addition, this thesis and [Barf, 2014] are to be consulted for detailed explanations.

3.3 HorizonSensorTest Software

The purpose of the HorizonSensorTest environment is to provide a convenient way to test and validate the HorizonSensor software on a desktop computer. It was implemented in Barf [2014] and its core was left untouched within this thesis work. Some functionalities to visualize all used pixels, all distorted and undistorted lines, the pixels selected by the outlier detection and a function to draw a general conic section were added. Furthermore, the functionality to forward custom arguments given at startup and a function to read Keyhole Markup Language (KML) files was implemented. The transformation chain described in Section 4.3.3 was also translated to `C++` code and integrated into the HorizonSensorTest environment. As thoroughly described in Barf [2014], the environment consists of two classes, the `TestBench` and the `TestCase`. The `TestCase` represents an abstract test and provides functions to analyze and visualize the horizon detection and attitude determination process. In order to create an actual test routine, a class that inherits from `TestCase` must be created and the method `run()` overwritten. During this thesis work, three different test cases have been implemented: the `StepsFullTest`, `RexusTest` and `SunSensorTest`. The `StepsFullTest` feeds the HorizonSensor with simulated image data created with GEP, displays every single step of the process and records the results together with the ground-truth. The `RexusTest` was implemented in order to analyze and optimize the robustness off the algorithm with real image data. It also displays every step and logs the results but without ground-truth because it does not exist for this set of data. The `SunSensorTest` is an attempt to find a configuration for the HorizonSensor to detect the Sun instead of the Earth and determine the relative attitude to the Sun. More information about the tests can be found in Section 4.5. HorizonSensorTest is programmed in `C++` and requires the libraries Boost and OpenCV to be installed.

3.4 Camera Simulator

The camera simulator is a software that runs on a desktop computer and emulates a camera for the embedded system. This software was implemented because the selection and implementation of a camera is not part of this thesis but the implementation of the HorizonSensor software must be validated and analyzed. With the camera simulator, any previously prepared movie can be fed to the embedded system and the results monitored and recorded. The camera simulator is implemented in `C++` with the Qt toolkit using the Qt-Creator IDE on Ubuntu 14.04. The communication is realized via the ethernet interface.

3.4.1 Communication Packages

There are six different communication packages to enable data exchange between the camera simulator and the embedded system. Communication packages can be broken down into two types: commands and data packages. Commands contain only a series of constant values in order to identify them, data packages contain additional data. There are three different commands: the `GetImageCommad`, `GetParamCommad` and `TerminateConnectionCommand`. The first one signals the receiver to send an `ImagePackage` and can only be send by the embedded system. The `GetParamCommad` is send by the camera simulator only and signals the embedded system to send the current parameters. The `TerminateConnectionCommand` is also only send by the camera simulator and signals the embedded system that the connection is terminated. Data packages are: the `ImagePackage`, `ResultPackage` and `SetParamPackage`. Latter contains the full set of parameters of the HorizonSensor and can be send and received by both, the camera simulator and the embedded system. The `ResultPackage` contains all results of a horizon detection and attitude determination and is send by the embedded system only. The `ImagePackage` contains the image data of a picture and is only send by the camera simulator. Since the communication is realized with the TCP/IP protocol, the transmission error correction is applied by the ethernet controller, hence the transmission can be assumed error-free.

3.4.2 Graphical User Interface

The graphical user interface (GUI) of the camera simulator is divided in three sections (cf. Fig. 3.2). The leftmost part provides control of the parameters. The currently used set of parameters can be requested from the embedded system and displayed in the lower part of the section. There, parameters may be changed and sent back to the embedded system. There is also the option to save and load the parameters to

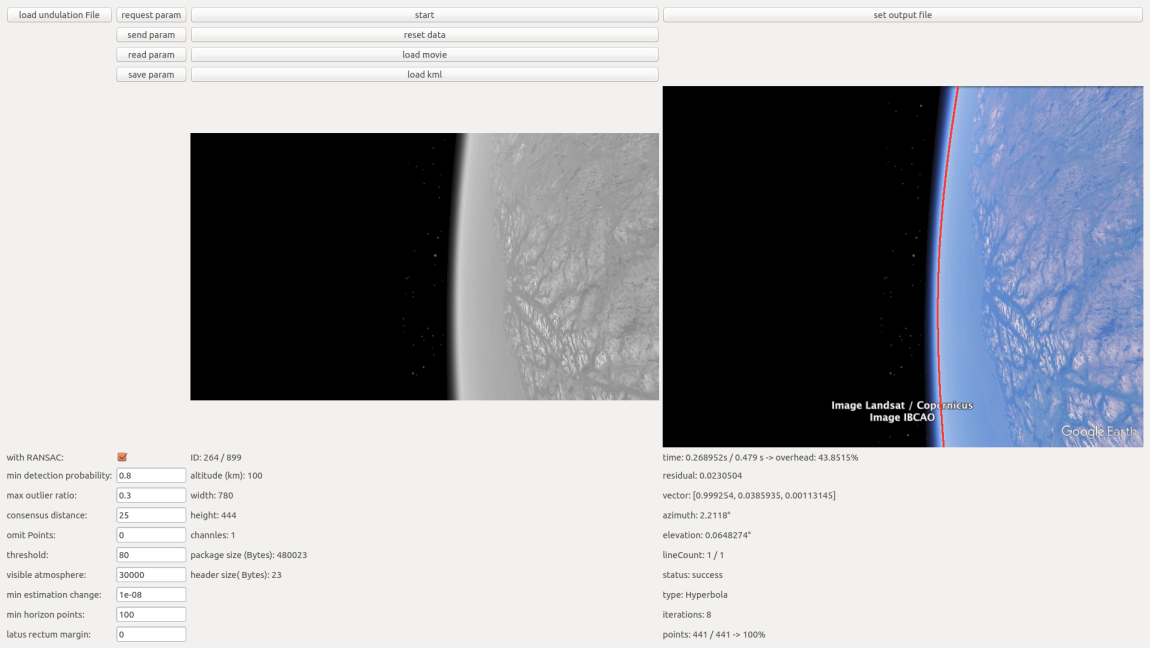


Figure 3.2: CameraSimulator GUI

and from an Extensible Markup Language (XML) file on the computer. The column in the center displays and controls what image data is sent to the embedded system. It provides functions to load movie and KML files (cf. Sec. 4.3.1). From here, the connection to the embedded system can be started and severed. The image that was actually sent and its characteristics are displayed in the center and lower part. In the right part of the GUI the results received from the embedded system can be monitored. It displays the original image (not transmitted by the embedded system) with the resulting conic section painted into it. In the lower part, all results are listed in detail. Moreover, a function to save the received results to a file on the desktop computer is provided.

3.5 Calibration Tool Software

The calibration tool software was developed in order to determine the intrinsic parameters of the optical system. The model of the optical system and the meaning of the parameters is explained in detail in Section 2.7. The described model is also implemented in the calibration toolbox of the computer vision library OpenCV. It is available within the `fisheye` namespace. OpenCV provides functionalities to determine the intrinsic parameters by analysing pictures of a chessboard pattern. Those patterns provide easily detectable feature points with known relative positions. By comparing the detected positions in the image with the known relative positions it

is possible to determine the intrinsic parameters. The actual process is a fit of the parameters to the measured data using the LMA. The pictures must be taken with the optical system that is to be calibrated while the focal length is constant. The calibration is only valid for this focal length.

The calibration tool is a command line program and takes videos where chessboard patterns are visible as input. It detects the pattern in every 60^{th} frame autonomously, determines the intrinsic parameters, saves the parameters to a file and is able to undistort full videos using these parameters. The tool is started with one mandatory (`<folder>`) and one optional (`<video>`) argument (cf. Lst. 3.3).

Listing 3.3: CalibrationTool Usage

```
./CalibrationTool <folder> [<video>]
```

All video files contained in `<folder>` are included in the calibration process. The video specified in `<video>` is undistorted. The detection algorithm takes a lot of time if there is no pattern in the picture to detect. In order to accelerate the process, the time intervals where the pattern is fully visible should be manually provided to the tool. This is done by saving one `<name>.times` file for every video in `<folder>` where `<name>` is the name of the video file. The content of the file is shown in Listing 3.4.

Listing 3.4: Structure of `.times` Files

```
<start> <end>
.
.
.
<start> <end>
```

`<start>` and `<end>` in each row is the start and end time in seconds of the period of the video the pattern is fully visible. For every period there must be a row. A XML file containing the intrinsic parameters and the undistorted video is saved in `<folder>`.

CHAPTER 4

Evaluation

In contrast to several other works in the field of horizon sensing with image processing where the absence of ground-truth debilitates the meaningfulness of the analysis, this thesis pays special attention to the comparison of true and measured attitude information. This chapter therefore explains in detail how test data, including its ground-truth, is created and how this data is then used to evaluate the accuracy. In addition, the robustness of the algorithm towards disturbances in the image is analysed using real footage from a flight on a sounding rocket. Furthermore, the execution time of the algorithm on the target platform is thoroughly studied. In addition, a theoretical runtime analysis of the algorithm and a theoretical evaluation of the observability of the different conic section types is performed.

4.1 Observability of Conic Section Types

The three possible conic section types ellipse, parabola and hyperbola are not observable at every position. There are systematic concepts that prevent the observation of certain types in dependance of the altitude and the field of view. This is an evaluation of the observability of conic sections.
In Section 2.5 it was derived that the projected curve is an hyperbola if the image plane intersects the central body. Furthermore, the horizon can only be detected if it is in the field of view. This implies, that there are combinations of FOV and altitude where there are no hyperbolae or only hyperbolae observable. The first case is illustrated in Figure 4.1. There the central body is fully inside the half room with positive $\vec{z}_c|_c$ coordinates but touches the image plane. This arrangement would cause a parabola as projection if the horizon is in field of view. In this case however, the conic FOV also only touches the earth. In order to get the horizon into the field of

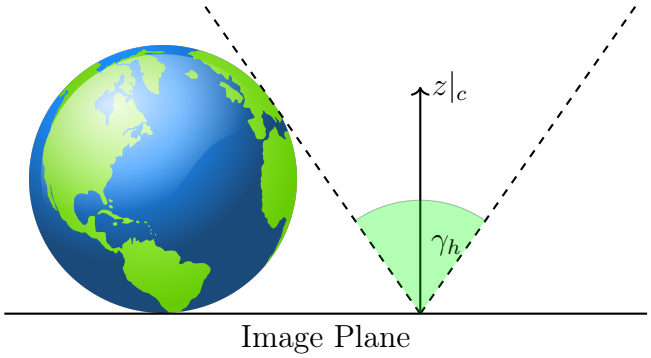


Figure 4.1: Configuration that does Not Allow Hyperbolae

view, the camera must be rotated towards the Earth. But that would cause the image plane to rotate away from the central body and thus makes a hyperbola impossible. The second case is illustrated in Figure 4.2. Here the image plane is in the same

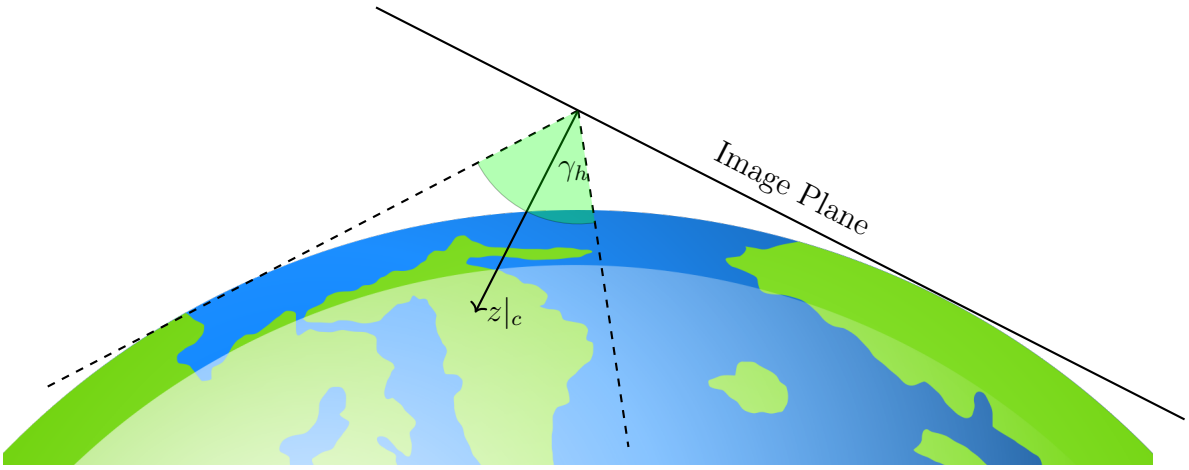


Figure 4.2: Configuration that only Allows Hyperbolae

configuration with the central body as in the first case but the camera is much closer to it. The FOV in this case is just not large enough to include the horizon. In order to get the horizon into the FOV the camera must be rotated away from the central body. This however would cause the image plane to intersect the central body and cause a hyperbola.
Both of these cases are edge cases for the parameters altitude and FOV. In the graph in Figure 4.3 the observable types are plotted against those two parameters. In addition, the area in which the whole central body is observable is marked red. The green area, where only hyperbolae are observable occurs only for small altitudes up to 2600km and small field of views. The blue area, where no hyperbolae are observable starts at the same altitude and grows fast up to a FOV 140° at a geostationary orbit.

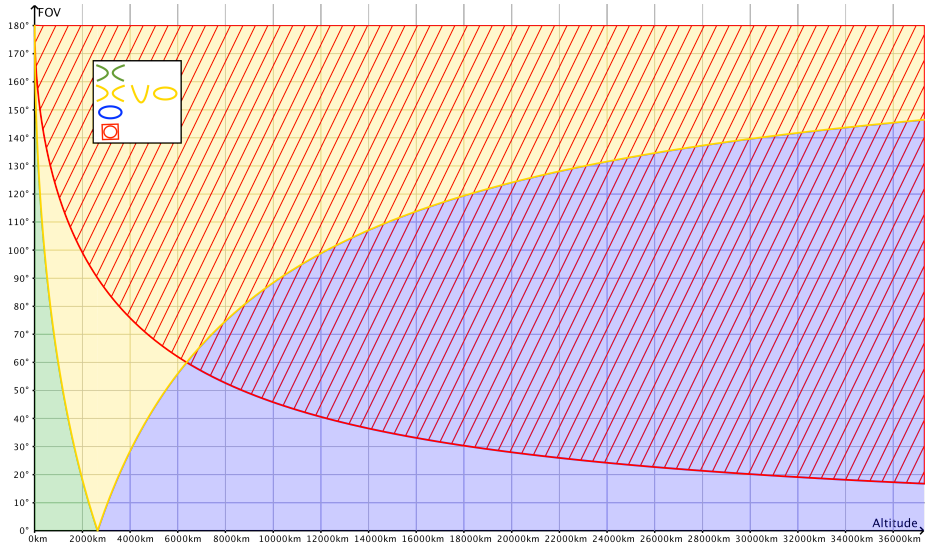


Figure 4.3: FOV and Altitude Configurations where the Projections: Hyperbola (H), Parabola (P) and Ellipse (E) can be Observed (Green: H, Yellow: H/P/E, Blue: E). In the Red Hatched Area it is Possible to Observe the Full Earth Disk.

The yellow area, where all projections are observable, fills the void between the two other areas. Although the full central body is hardly observable in lower altitudes it is very easily observable with a FOV down to 20° in the geostationary orbit. For the application on sounding rockets however, altitudes higher than 800km are of no interest. The graph in Figure 4.4 is an excerpt of Figure 4.3 with the relevant altitudes up to 800km to analyse the possible observations in detail. Here the hyperbola-only

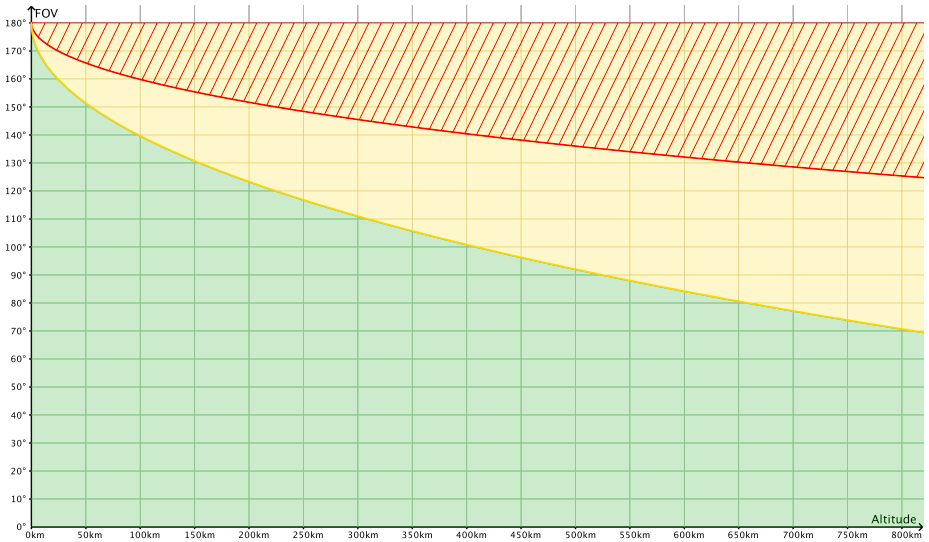


Figure 4.4: Relevant Excerpt of Figure 4.3 for Sounding Rockets (same Color Code)

area clearly dominates the graph. For FOVs lower than 60° only hyperbolae can be observed and for FOVs lower than 120° the central body cannot be fully observed. This graph shows the need of an sensor that is able to cope with hyperbolae and ellipses if it is to provide reliable attitude information.

4.2 Runtime Analysis

The theoretical runtime analysis of the algorithm helps to find the most computational expensive parts of the algorithm which might need to be optimized. It also reveals the magnitude of the impact of the parameters. The analysis, starting with the most often called method `CalcVec`, is written in the Landau-Notation. The runtime t_v of the vector estimation is measured in arithmetical operations. The input of the method is a number of points with the magnitude p_v. It has two nested `for` loops. The outer loop repeats maximally e (=const.) times and the inner loop p_v times. The inner loop conducts 51 arithmetical operations, thus in total there are $51ep_v$. In Landau-Notation this is written as

$$t_v \in \mathcal{O}(51ep_v) = \mathcal{O}(p_v) \tag{4.1}$$

and constitutes a linear runtime in the number of sample points p_v.

This method is called multiple times by the function `ransac`. It receives a number p_r of points and has one `for` loop that is repeated r (=const.) times (RANSAC iterations). In this loop, the function `CalcVec` is called each iteration to estimate the vector using three random points. Calling `CalcVec` r times with $p_v = 3$ points contributes with $\mathcal{O}(3 \cdot 51 \cdot r \cdot e)$ to the total runtime of the `ransac` function. Later in the loop, the number p_c of points that support the model are counted which contributes with $\mathcal{O}(p_r)$ each iteration and thus with $\mathcal{O}(r \cdot p_r)$ in total. After the loop finishes the model is estimated with all supporting samples p_c. This step contributes with $\mathcal{O}(51ep_c)$ to the total runtime of `ransac`. Since the set of supporting points is a subset of all points, p_c can be approximated with the upper bound $p_c \leq p_r$. To calculate the residual of the estimation, the distance of every supporting point (p_c) is determined, which costs $\mathcal{O}(p_c)$. Therefore, the total runtime of the RANSAC algorithm in Landau-Notation is

$$\begin{aligned} t_r \in \quad & \mathcal{O} \quad (3 \cdot 51 \cdot r \cdot e + r \cdot p_r + e \cdot p_c + p_c) = && (4.2)\\ & \mathcal{O} \quad (r \cdot e + r \cdot p_r + e \cdot p_r + p_r) = && (4.3)\\ & \mathcal{O} \quad ((r + e + 1)p_r) = && (4.4)\\ & \mathcal{O} \quad (p_r) && (4.5) \end{aligned}$$

and hence is linear in the number of input points p_r.

The topological search finds a number P of edge pixels in an image with height h and width w where the latus-rectum of the projected curve of the Earth is $2l$. The grid search analyzes all pixels at the image border and additionally those in certain horizontal rows (cf. Sec. 2.6). The search at the image borders contributes with $\mathcal{O}(2h + 2w)$ and the horizontal lines with $\mathcal{O}(\frac{hw}{2l})$ to the total runtime of the edge detection. For every new edge that is found, the border following algorithm is started. In total this procedure finds $|L|$ different lines. The i-th line L_i contains a number of p_i points. Thus $\sum_{i=0}^{|L|} p_i = P$ is the total amount of found edge pixels. The i-th border following process iterates through p_i points, thus all executions of the border following process iterate through P pixels in total. Hence the contribution of the border following process is $\mathcal{O}(P)$.
The HorizonSensor provides the function to omit points in order to decrease the processing time (cf. Sec. 4.5.2). The number p_a of points that are not ignored and further processed is $p_a = \frac{P}{1+o}$ where o is the number of omitted points. The upper limit for p_a is P, which is the case if $o = 0$
The next step is to undistort and transform the points. This process requires an iterative process to numerically approximate θ_u (cf. Sec. 2.7). This loop iterates u (=const.) times and has a constant number of arithmetical operations per iteration. The undistortion therefore contributes with $\mathcal{O}(u \cdot p_a)$.
For every found line L_i the RANSAC algorithm is executed which costs $\mathcal{O}\left(\sum_{i=0}^{|L|} t_r(\frac{p_i}{1+o})\right)$ in total. The runtime of the edge detection is thus

$$t_t \in \mathcal{O}\left(2h + 2w + \frac{hw}{2l} + P + up_a + \sum_{i=0}^{|L|} t_r\left(\frac{p_i}{1+o}\right)\right) = \tag{4.6}$$

$$\mathcal{O}\left(P + up_a + \sum_{i=0}^{|L|} t_r\left(p_i\right)\right) = \tag{4.7}$$

$$\mathcal{O}\left(P + uP + P\right) = \tag{4.8}$$

$$\mathcal{O}\left((1 + u + 1)P\right) = \tag{4.9}$$

$$\mathcal{O}\left(P\right) \tag{4.10}$$

and therefore linear in the number of edge pixels. The sum can be replaced with P because the sum of the runtimes of many subsets of one set equals the runtime of the whole set at once. Realtime systems require a upper limit of the runtime. Here, the upper limit of edge pixels can be approximated with the total number of pixels in the image. Thus, the upper limit of the runtime is

$$t_t \leq \mathcal{O}(wh) \tag{4.11}$$

4.3 Creating Simulated Test Data

In order to determine attitude measurement errors, the true (reference) attitude must be known. This reference data was created using the GEP program. It allows to create footage that a camera with given attitude, position, image dimensions and field-of-view would produce. The simulation includes stars, the Sun and a photorealistic illustration of the atmosphere.

4.3.1 Creating KML Files

GEP can read so called KML files which follow the XML syntax and contain all needed information for a series of views in GEP. Such a series of views is called a `Tour` and contains the element `Playlist` which is the parent element of multiple `FlyTo` elements. Each `FlyTo` element represents one view and contains all parameters given in Table 4.2. There is a Matlab toolbox called *KML toolbox* available in the

Name	**Symbol**	**Description**
longitude	λ	The geodetic longitude of the camera position
latitude	ϕ	The geodetic latitude of the camera position
altitude	H	The altitude above MSL of the camera
heading	θ_h	The angle of the first rotation of the $zx'z''$-rotation from ENU system to body fixed system in negative direction.
tilt	θ_t	The angle of the second rotation of the $zx'z''$-rotation from ENU system to body fixed system in positive direction.
roll	θ_r	The angle of the third rotation of the $zx'z''$-rotation from ENU system to body fixed system in positive direction.
horizontal FOV	γ_h	The full angle that determines the FOV in the horizontal plane.

Table 4.2: The available Information from GoogleEarth

Matlab File Exchange platform which allows to create KML files with an automated Matlab script. Such a script was already available at the department of RFT of the GSOC and was therefore, apart from minor modifications, not part of this thesis work. However, it was used to create KML files which produced desired views of the Earth.

4.3.2 Creating Movies

As already mentioned, GEP can read KML files and render realistic views accordingly. For this application it is important to deactivate all options that draw additional information into the image like country borders, roads or labels. Only the Terrain option and the option *Use photo realistic atmosphere rendering (EXPERIMENTAL)* in the preferences should be activated. All movies were created with the build-in Movie Maker and the settings

- Compression Type: H.264
- Frames per Second: 30
- Key Frame Option: Deactivated
- Limit Data Rate Option: Deactivated
- Image Size: 800x600
- Compression Quality: Best

and saved in the .mov format.

4.3.3 Coordinate System Transformations

The KML files can also later be read to extract the true attitude (the ground-truth) but that requires a sequence of coordinate transformations. Figure 4.5 illustrates the sequence of the transformations which is explained below.

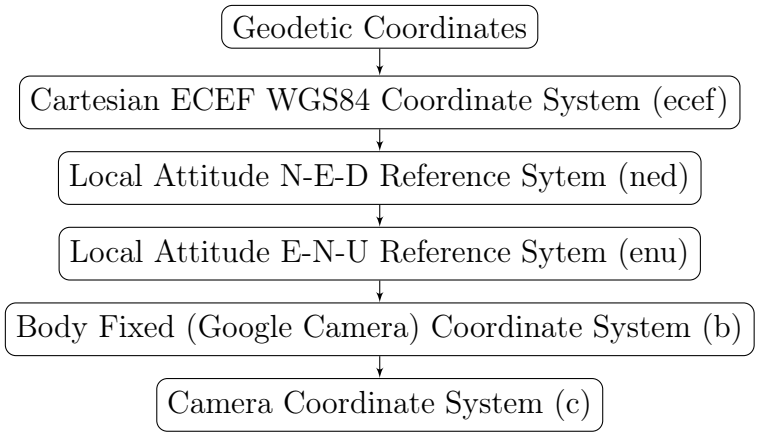


Figure 4.5: Flow Diagram of Coordinate Transformations

Geodetic to Earth-Centered-Earth-Fixed

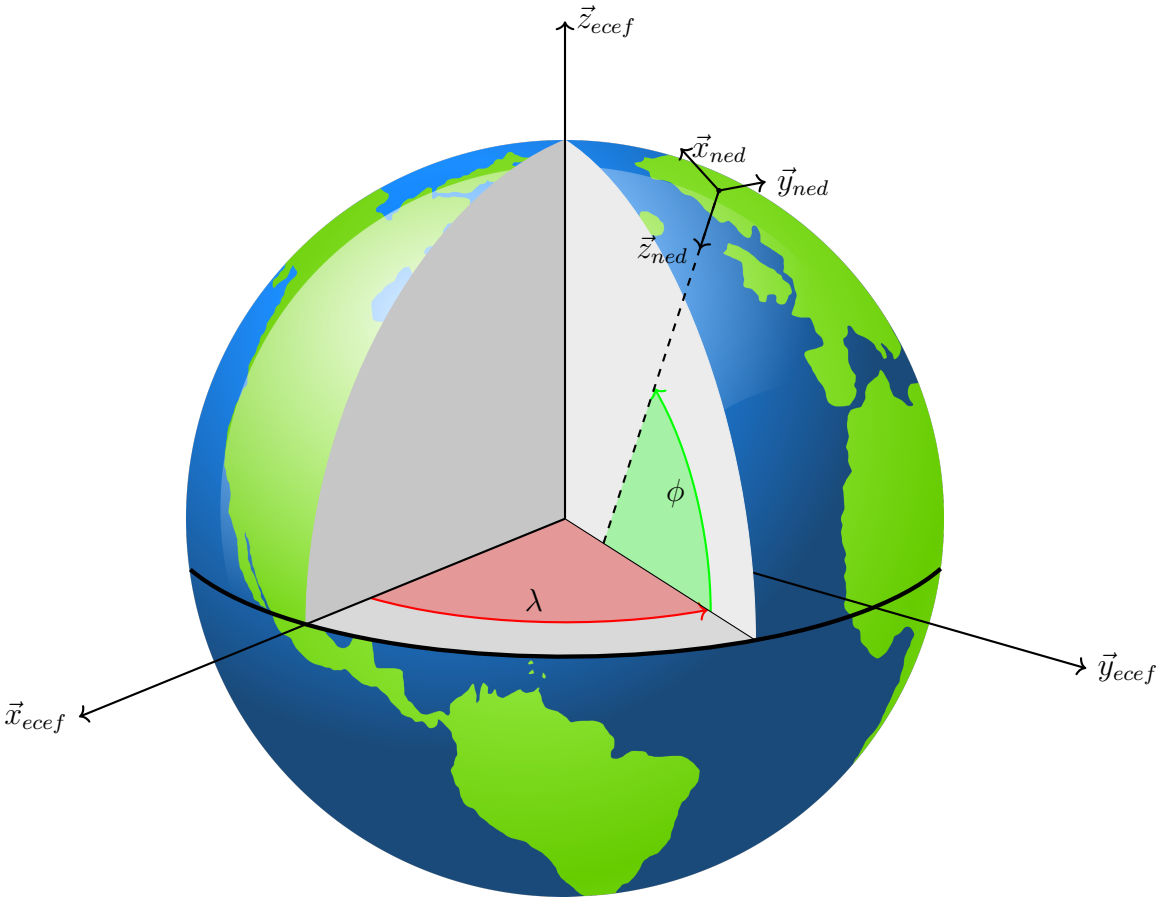


Figure 4.6: ECEF and NED Coordinate System

The geodetic position must first be converted into cartesian coordinates. The chosen coordinate system is an Earth centered Earth Fixed (ECEF) coordinate system defined in the technical report of the World Geodetic System 1984 (WGS84) [cf. WGS, 2000, sec. 2.1]. Its origin is at the Earth's centre of mass. Its $\vec{z}_{ecef}$-axis is in direction of the International Earth Rotation Service (IERS) Reference Pole (IRP). The x-axis is the intersection of the IERS Reference Meridian (IRM) and the plane passing through the origin and normal to the Z-axis. The y-axis completes a right-handed orthogonal coordinate system. The geodetic system regards the Earth as geoid whereas the ECEF system sees the Earth as ellipsoid of revolution. The geoid allows a better approximation of the actual mass distribution. The geodetic altitude H is given relative to the geoid's surface or, in other words, above mean sea level (MSL). That figure is highly dependent on longitude and latitude and varies strongly. There is no easy way to analytically describe these variations sufficiently well but there is a difference to the altitude normal h to the ellipsoid of revolution that cannot be ignored. For an altitude H relative to the geoid the height relative to the ellipsoid of revolutions

$$h = H + U(\lambda, \phi) \tag{4.12}$$

where $U(\lambda, \phi)$ is an undulation depending on longitude and latitude [cf. WGS, 2000, sec. 6.1]. U is given by a look-up table as published by the US National Geospatial-Intelligence Agency (NGA) in their Earth Gravitational Model from 2008 (EGM2008). It is available either as a 1x1-minute or as a 2.5x2.5-minute grid. The values in between the grid raster result from an interpolation.
Let λ be the geodetic longitude, ϕ the geodetic latitude, a the semi-major axis and e the first eccentricity of the ellipsoid of revolution. Then the cartesian coordinates of the origin of the NED reference system in the WGS84 ECEF system are

$$\vec{p}_{ecef} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}_{ecef} = \begin{bmatrix} (N(\phi) + h)\cos\phi\cos\lambda \\ (N(\phi) + h)\cos\phi\sin\lambda \\ ((1 - e^2)N(\phi) + h)\sin\phi \end{bmatrix} \tag{4.13}$$

where the normal curvature radius

$$N(\phi) := \frac{a}{\sqrt{1 - e^2 \sin^2\phi}} \text{ [cf. WGS, 2000, eqn. 4-14/4-15]} \tag{4.14}$$

and the first eccentricity

$$e := \sqrt{d(2 - d)} \tag{4.15}$$

in terms of the flattening d. The WGS [2000] defines the semi-major axis $a = 6378137.0$m and the reciprocal of flattening $1/d = 298.257223563$.

Earth-Centered-Earth-Fixed to North-East-Down

The north-east-down (NED)-system's $\vec{x}_{ned}$-axis is oriented towards the north pole, the $\vec{y}_{ned}$-axis points to the east and the $\vec{z}_{ned}$-axis aligns with the normal to the geoid and completes a right handed orthogonal coordinate system as illustrated in Figure 4.7. A conversion into this system is not necessarily needed for this application but since it is very popular in aircraft avionics there is literature and an implementation available which simplifies some of the work. The transformation from this system to the actually necessary one (east-north-up (ENU)) is then very simple (see Sec. 4.3.3). The rotation of the ENU system relative to the ECEF system can be described with a rotation around the $\vec{z}_{ecef}$-axis with the angle λ and a subsequent rotation around the new $\vec{y'}_{ecef}$-axis with the angle $-\pi/2 - \phi$. The rotation matrix is

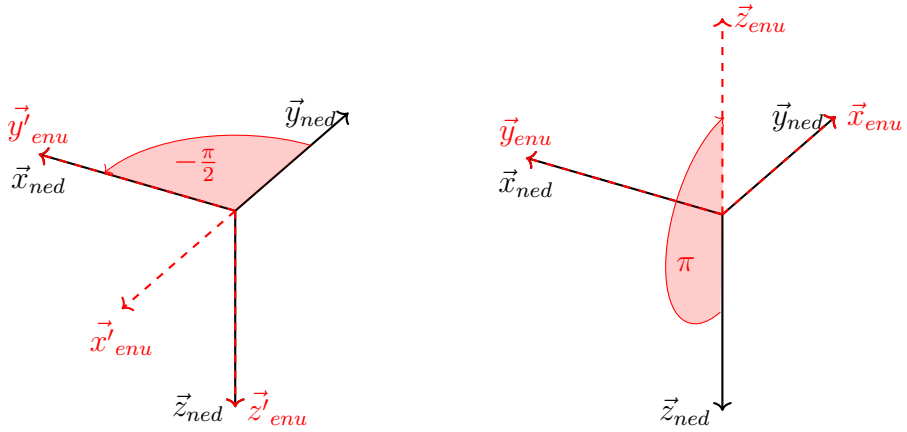
$$R|_{ned}^{ecef} = R_z(\lambda) \cdot R_{y'}\left(-\frac{\pi}{2} - \phi\right) = \begin{bmatrix} -\cos\lambda\sin\phi & -\sin\lambda & -\cos\lambda\cos\phi \\ -\sin\lambda\sin\phi & \cos\lambda & -\sin\lambda\cos\phi \\ \cos\phi & 0 & -\sin\phi \end{bmatrix}. \quad (4.16)$$

North-East-Down to East-North-Up

ENU describes the orientation of the axis similarly to the NED system in the last section. The $\vec{x}_{ned}$-axis of this system points to the east, the $\vec{y}_{ned}$-axis points to the north pole and the $\vec{z}_{ned}$-axis aligns with the normal to the ellipsoid of revolution and completes a right handed orthogonal coordinate system. As illustrated in Figure 4.7 the transformation from the NED system to the ENU system is a constant rotation. Here it was achieved by a rotation around the z_{ned}-axis with the angle $-\pi/2$ and a subsequent rotation around the new $\vec{y'}_{enu}$-axis with the angle π. This leads to the matrix in Equation 4.17. Since it is a symmetrical matrix it also describes the transformation in the opposite direction.

$$R|_{enu}^{ned} = R_z\left(-\frac{\pi}{2}\right) \cdot R_{y'}(\pi) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} = R|_{ned}^{enu} \quad (4.17)$$

As expected the axes $\vec{x}_{ned}$ and $\vec{y}_{ned}$ coincide and point to the north pole, the axes $\vec{y}_{ned}$ and $\vec{x}_{enu}$ coincide and point to the east and the axes $\vec{z}_{ned}$ and $\vec{z}_{enu}$ point in opposite directions, down and up. With Equations 4.16 and 4.17 the transformation from the ECEF to the ENU system is given by



(a) First Rotation (b) Second Rotation

Figure 4.7: Rotations from NED to ENU System

$$R|_{enu}^{ecef} = R|_{ned}^{ecef} R|_{enu}^{ned} = \begin{bmatrix} -\sin\lambda & -\cos\lambda\sin\phi & \cos\lambda\cos\phi \\ \cos\lambda & -\sin\lambda\sin\phi & \sin\lambda\cos\phi \\ 0 & \cos\phi & \sin\phi \end{bmatrix}. \tag{4.18}$$

East-North-Up to Body Fixed (Google Camera)

The rotation from the ENU reference frame to the body fixed coordinate system is given by three Euler angles for the rotation sequence $zx'z''$. The first rotation angle is called the heading θ_h, the second rotation angle is called tilt θ_t and the third and last rotation angle is called roll θ_r. To make things more complicated, Google defines the heading, the first rotation angle, as a rotation in the opposite direction. That is why a new symbol θ_h^- is defined as $\theta_h^- := -\theta_h$. The rotation from the ENU reference frame to the body fixed coordinate system is then defined as

$$R|_b^{enu} = R_z(\theta_h^-) R_{x'}(\theta_t) R_{z''}(\theta_r) \tag{4.19}$$

and can be written as a single matrix as seen in Equation 4.20.

$$R|_b^{enu} =$$

$$\begin{bmatrix} \cos\theta_r\cos\theta_h^- - \sin\theta_r\cos\theta_t\sin\theta_h^- & \sin\theta_r\cos\theta_h^- + \cos\theta_r\cos\theta_t\sin\theta_h^- & \sin\theta_r\sin\theta_h^- \\ -\cos\theta_r\sin\theta_h^- - \sin\theta_r\cos\theta_t\cos\theta_h^- & -\sin\theta_r\sin\theta_h^- + \cos\theta_r\cos\theta_t\cos\theta_h^- & \sin\theta_t\cos\theta_h^- \\ \sin\theta_r\sin\theta_t & -\cos\theta_r\sin\theta_t & \cos\theta_t \end{bmatrix} \tag{4.20}$$

Body Fixed (Google Camera) to Camera

The only difference between the body fixed coordinate system and the camera coordinate system in this application is a rotation of the angle π around the $\vec{x}_b$-axis as illustrated in Figure 4.8.

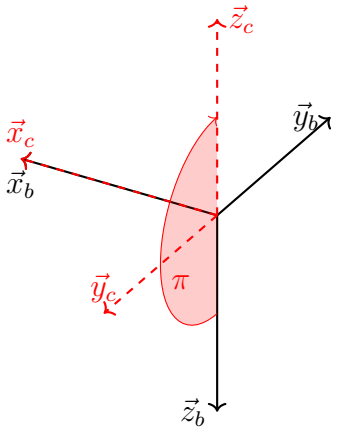


Figure 4.8: Rotation between the Body-Fixed and the Camera System

This rotation is described as matrix in Equation 4.21. As it is a symmetrical matrix, it also describes the rotation from the camera system to the body fixed system.

$$R|_c^b = R_x(\pi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} = R|_b^c \tag{4.21}$$

Combining Equation 4.20 and Equation 4.21 results in a matrix describing the rotation from the ENU reference frame to the camera frame as written in Equation 4.23.

$$R|_c^{enu} = R|_b^{enu} R|_c^b = \tag{4.22}$$

$$\begin{bmatrix} \cos\theta_r \cos\theta_h^- - \sin\theta_r \cos\theta_t \sin\theta_h^- & -\sin\theta_r \cos\theta_h^- - \cos\theta_r \cos\theta_t \sin\theta_h^- & -\sin\theta_r \sin\theta_h^- \\ -\cos\theta_r \sin\theta_h^- - \sin\theta_r \cos\theta_t \cos\theta_h^- & \sin\theta_r \sin\theta_h^- - \cos\theta_r \cos\theta_t \cos\theta_h^- & -\sin\theta_t \cos\theta_h^- \\ \sin\theta_r \sin\theta_t & \cos\theta_r \sin\theta_t & -\cos\theta_t \end{bmatrix}$$

The the results from HorizonSensor algorithm are also given in this camera coordinate system (cf. Sec. 2.4.1). Thus, with the described transformations, the attitude information given in the KML files can be compared with the result from the

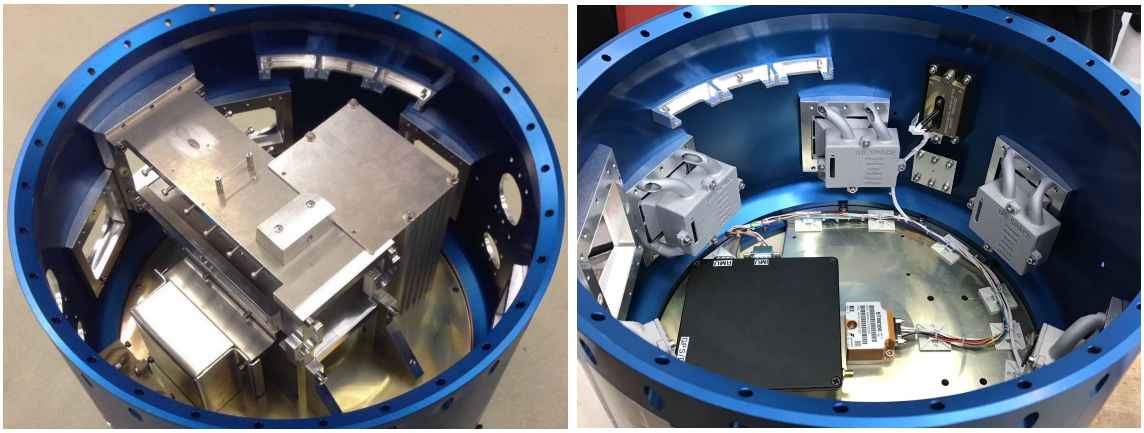
HorizonSensor.

4.4 Real Footage

Prooving the algorithms functionality with simulated data is not satisfactory for a real application because even the best simulation can only approximate reality but never perfectly replicate it. In this case the illustration of the atmosphere and the influence of the sun on the atmosphere and the optical system of the camera can hardly be modelled accurately, especially not within the scope of this work. Therefore, tests with real footage must be conducted.

4.4.1 UB-SPACE Experiment Description

On March 15th, 2017 the sounding rocket REXUS 21 was launched from ESRANGE Space Center in Kiruna, Sweden. On board of that rocket was the student experiment UB-Space from the University of Bremen. It carried six GoPro cameras that were arranged equiangular around the experiment module to observe the outer environment (cf. Fig. 4.9b). Also inside the module there was a cube with an edge length of 60mm. This cube was ejected with an ejection system after despin had taken place. The ejection mechanism can be seen in Figure 4.9a.
This experiment aims to reconstruct the cubes movement as it drifts away by analysing the footage from the GoPro cameras. Those cameras have a very high dynamic range,



(a) Experiment with Ejection System (b) Experiment with GoPros and IMU

Figure 4.9: UB-Space Experiment Module [cf. UB-SPACE]

are able to cope with strong vibrations and provide videos with high resolution of 1080p while still maintaining a frame rate of 60 frames per second. Due to their fisheye lenses however, they introduce strong distortions to the picture. Fortunately, it was possible to take calibration photos before and after the flight that are used to calculate a compensation for the distortions. In addition to this experiment, an IMU and a set of sun sensors had been put into the module. Thanks to the close cooperation with the DLR and the student team, all sensor data is now available for this thesis work. The flight went nominally and the experiment worked almost perfectly. Only one camera shut down during ascent and one camera turned off during reentry. Apart from that, the GoPros took stunning videos of the Earth in altitudes from 0 up to 84 km. The sun sensor and IMU also worked nominally. The footage from the six cameras is used to determine if the algorithm works not only with simulated but also with real images. Unfortunately, the accuracy of the IMU together with the sun sensors is not sufficient to determine the accuracy of the horizon sensor. But the two-axis attitude information calculated by each horizon sensor camera was fused with the attitude information from the other two sensors and thus contributes to a more accurate full attitude information. The data fusion, however, is not part of this work but the results are available for analysis of the algorithm performance.

4.4.2 Calibration

As already mentioned, the GoPro cameras introduce high distortions to the picture and therefore must be calibrated. Calibration pictures, as seen in Figure 4.10, have been taken before the flight and have been processed with the tool described in Section 3.5. In this case, the pattern with 5x8 inner corners was printed on a DIN A4 sized paper and just laid on the floor in front of the camera while the camera was moved. Although the GoPro cameras are all of the same model it is not guaranteed that the

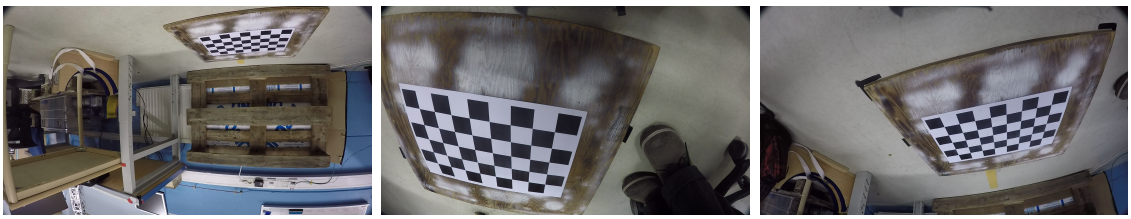


Figure 4.10: Calibration Photos of Chessboard Patterns

calibration parameters are identical since manufacturing and assembly processes are not perfect. Of course, also the calibration itself is not perfect and may introduce some noise. In Table 4.3 the distortion coefficients for all six cameras are listed. In general the values are considered consistent with the assumption that the cameras

are very similar to each other as they are approximately within the same order of magnitude. Table 4.4 shows the estimated parameters of the camera matrix. The

Cam	k_1	k_2	k_3	k_4
	$\times 10^{-3}$			
1	53.98	15.50	−9.942	2.174
2	57.34	2.366	4.032	−2.047
3	50.89	9.995	−3.917	0.7973
4	48.92	22.81	−19.23	6.771
5	56.69	7.286	−5.518	2.625
6	50.41	13.34	−5.537	0.3525

Table 4.3: Distortion Coefficients of the Pre-Flight Calibration

expected focal length in pixels without errors can be calculated with

$$f = f_x = f_y = \frac{w}{2\tan\left(\frac{\gamma_h}{2}\right)} \tag{4.23}$$

as derived in Section 2.5.1, where w is the width of the image in pixels and γ_h is the horizontal field-of-view. For $w = 1920$ and $\gamma_h = 94.4°$ [GoPro, 2017] the focal length

Cam	f_x	f_y	c_x	c_y
1	885	879	982	579
2	883	879	903	527
3	877	872	911	540
4	883	879	912	522
5	885	880	928	512
6	878	871	919	531

Table 4.4: Camera Matrix Parameters of the Pre-Flight Calibration

is $f = 888 \pm 2$ and the image center without errors of an image of the size 1920x1080 should be at $(960, 540)$. The estimated values for the focal length are slightly smaller for all cameras but are considered valid. The estimated image center seems to have a systematic error because all estimated centers but the one of the first camera are shifted in approximately the same direction. The plot in Figure 4.11 illustrates the positions of the image centers. The first camera was calibrated upside down, so if the estimated center of the second camera is inverted in the center point it is in the same area as the other estimated centers. This systematic error might be due to a unbalanced sample collection for the calibration. It might create a small offset to the resulting angles comparable to a mounting angle error.

Figure 4.12a shows an image the second camera took before lift off. It shows a

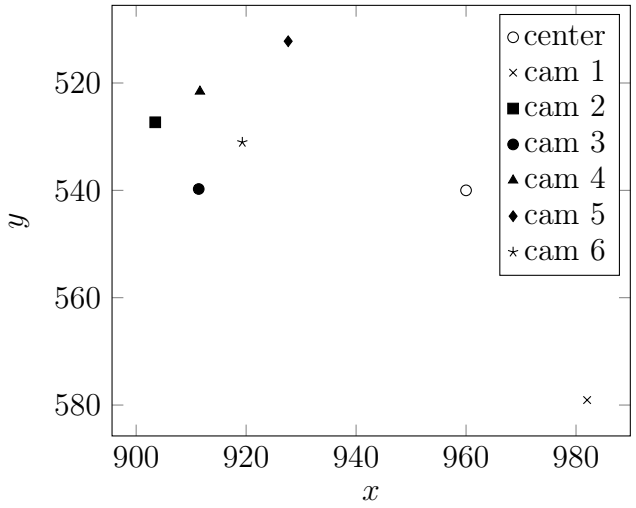
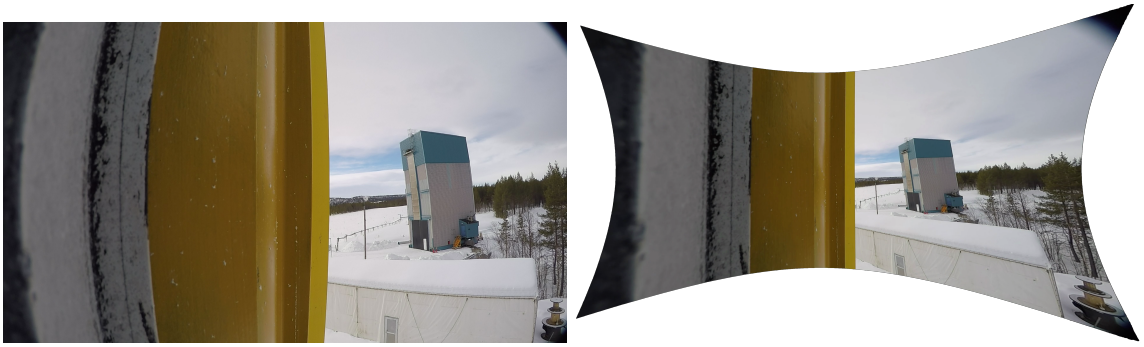


Figure 4.11: Plot of the Estimated Center Points

part of the rail of the launcher (yellow) and the MAXUS building. It can clearly be



(a) Distorted (b) Undistorted

Figure 4.12: Result of the Undistortion of a GoPro Image

seen that lines that are straight in reality are bend towards the borders of the image. Figure 4.12b shows the same image after undistortion by the calibration tool with the parameters above and the calibration parameters in the last section. Here all lines that are supposed to be straight, look indeed straight which is a sign for a good calibration. Any other distortions like the expansion of objects in the areas close to the image borders are not distortions due to the fisheye projection but result solely form the perspective view.

4.5 Results

With the test data explained in the last two sections the accuracy, robustness and sample time was determined in multiple experiments.

4.5.1 Sample Time

Commonly the term sample time T depicts the time it takes between two samples. The sample frequency is then given by $f_s = \frac{1}{T}$ and describes the rate at which samples are available. In this case the sample time has two parts, the time T_c necessary for communication and the time T_p it takes the algorithm to process the image. T_c is only dependent on the speed of communication and the amount of data which varies only with the size of the image. T_p on the other hand is highly dependent on a number of parameters. One of them is the system clock frequency which is kept constant at 75 MHz. The major factors influencing the processing time and their impact are ascertained in this evaluation.

Method

It must be noted that the horizon sensor runs as a thread of the realtime operating System RODOS and some of the processor time is always consumed by the kernel. Furthermore, the measurement accuracy of the time is limited by the processor's clock frequency. Here the uncertainty of the function `NOW()` which returns the current system time in nanoseconds is at least $\pm$100ns. Since it is called two times (start & stop time of a measurement) the uncertainty doubles in total.
The measurement of the process time T_p was started just before the `findHorizon()` function was called and stopped right after it returned. As test data, a simulated movie where the Earth horizon is always in view and rotates with constant elevation of about 0° and constant altitude of 100km around the principal axis was chosen. That means only the azimuth angle is changing. Every frame is cropped from a size of 800×600pixels to 780×444pixel to get rid of the shadow and text that was drawn into the image by GEP. This setup allows the analysis of a large variety of different horizon line sizes. For each frame the time dependency of three parameters were measured: the image dimensions, the number of RANSAC iterations and the number of omitted points. Therefore, all combinations of scale factors, RANSAC iterations and omitted points were measured for every frame. The range of scale factors was $[0.025; 1]$ with a step size of 0.025, the range of RANSAC iterations was $[0; 100]$ with a step size of 1 and the range of omitted points was $[0; 70]$ with a step size of 1. This results in $\frac{1}{0.025} \cdot 100 \cdot 70 = 280000$ measurements per frame and took about 1.5 days

per frame. Therefore, only three frames (where the horizon is horizontal, vertical and diagonal) have been processed.

Horizon Line Length

In Section 4.2 the runtime was found to be increasing linearly with the number of edge pixels. This should be reflected in actual measurements of the processing time with simulated image data. Since the simulated data is perfect, the only edge pixels are those of the horizon line. The runtime analysis only gives a qualitative proposition about the growth of the processing time whereas measurements provide quantitative information. Therefore, a measurement of the processing time on the target platform in dependancy of the horizon line length for simulated optical systems with different FOVs was conducted. The graph in Figure 4.13 illustrates the results. Apparently,

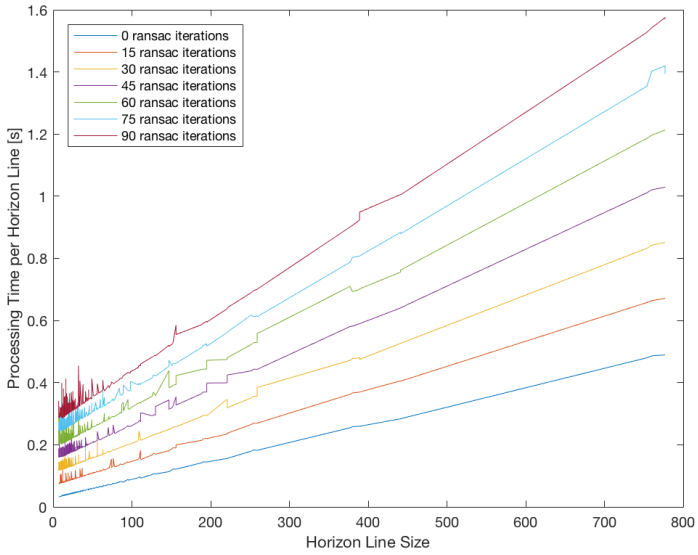


Figure 4.13: Processing Time vs. Line Size

the curves are almost straight lines, which supports the prediction. It can also be seen, that the number of RANSAC iterations has a major influence on the processing time.

RANSAC Iterations

To further analyse the effect of the number of RANSAC iterations on the processing time, the data of the same measurement is illustrated differently. The graph in

Figure 4.14 shows the processing time versus the number of RANSAC iterations for different line lengths. These plots represent straight lines as well which leads to the

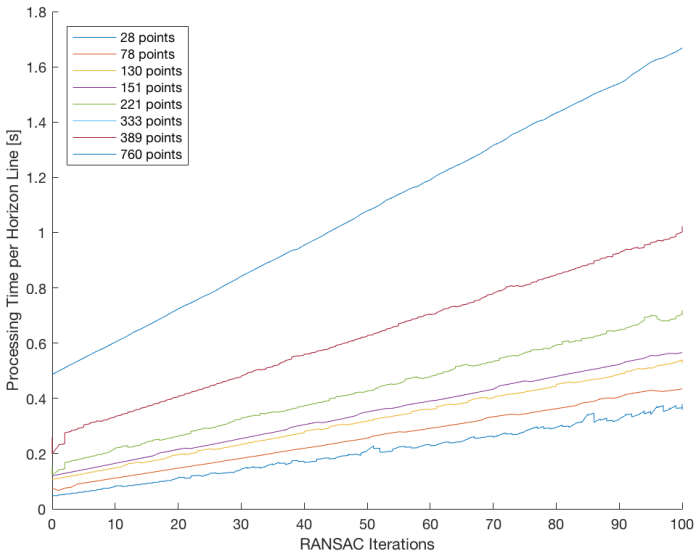
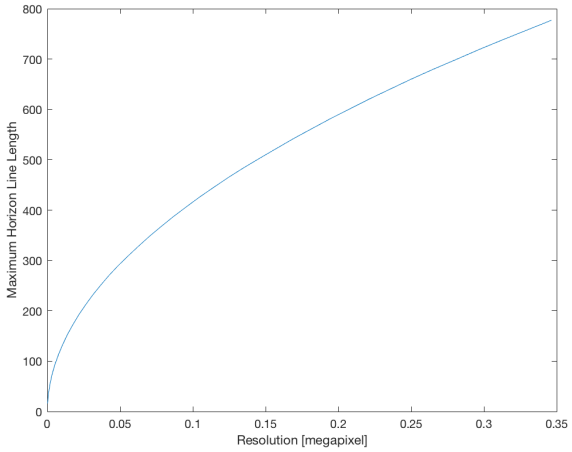


Figure 4.14: Processing Time vs. RANSAC Iterations

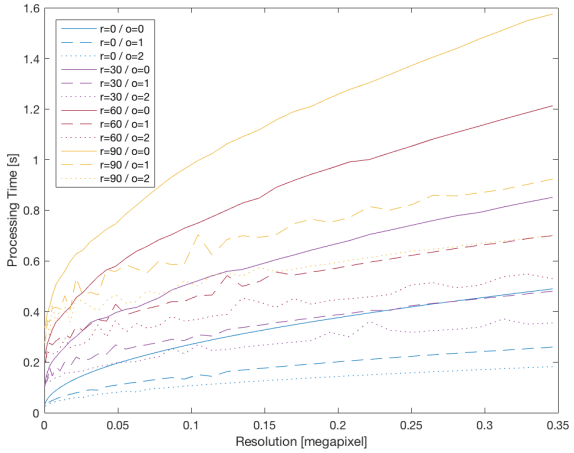
conclusion that the processing time is linearly dependent on the RANSAC iterations. This proposition is supported by the theoretical runtime analysis in Section 4.2.

Resolution

In this section the resolution of the image as influencing factor is investigated. The smaller the resolution the less pixels are contained in the horizon line which leads to a decreased processing time with respect to above measurement. The growth of the line length with increasing resolution is plotted in Figure 4.15a. The curve was measured for one particular line in one particular frame. Since the processing time is in linear dependency to the line length, the dependency of the processing time on the resolution should be a very similar curve as in this graph. The four solid curves in Figure 4.15b are plots of the processing time versus the resolution for different numbers of RANSAC iterations. One can see that those curves are indeed very similar which also endorses the validity of the theory. The dotted and dashed lines are explained later.



(a) Line Length vs. Resolution



(b) Processing Time vs Resolution

Kill Switch Reaction Time

As described in Section 3.2.2 the HorizonSensor has a kill switch implemented, which terminates the process while maintaining valid results. This termination cannot happen instantaneous because the current operation must be completed. The time it takes between the activation of the switch and the termination of the process is crucial for realtime systems. Therefore, a series of measurements were conducted in order to determine the impact on the embedded system. In order to achieve an activation of the kill switch from outside the HorizonSensor while it is running, another thread was implemented, the `HSInterruptor`. It has the sole purpose to activate the kill switch a certain time after the HorizonSensor process is started. The time is stopped between the moment right after the kill switch is activated and right after the HorizonSensor process terminates. This measurement is repeated for 900 different samples. The mean value of the reaction time was found to be 24.8ms with a variance of $0.22 \times 10^{-3}\text{s}^2$. This value must be considered when dealing with realtime systems. It must also be considered that the result may be significantly degraded. The kill-switch stops the grid search and the border following in the edge detection, it does not allow any new RANSAC or least-squares iteration to start. Thus, activating the kill-switch can have three negative effects on the quality of the results. Firstly, it may happen that the horizon line is not found. Secondly, the horizon line was found but the outliers were not removed correctly. Or lastly, the horizon line was found, the outliers were removed correctly but the calculated vector has a large offset.

Omitted Points

Thus, in order to reduce the processing time one can either reduce the resolution of the image or reduce the number of RANSAC iterations. But there is a third option:

omitting points. This option ignores a certain amount of points that are found during the border following process. In the implementation of the HorizonSensor the number of omitted points can be set as parameter. The non-negative number defines how many found pixels are ignored until one is accepted. If the number is zero no pixels are ignored, if it is one, 1/2 of all pixels are ignored, if it is two, 2/3 of all pixels are ignored and so on. Thus, a ratio of $\frac{1}{1+\mu_{op}}$ of all pixels is accepted, where μ_{op} is the number of omitted points. The dashed and dotted lines in Figure 4.15b are curves where μ_{op} is one and two respectively. Obviously, it is possible to reduce the processing time greatly by omitting points. Since the processing time is in linear dependancy of the line length and the line length is in reciprocal dependency of the omitted points, the dependency of the processing time on the number of omitted points must be described by a reciprocal function. To investigate if this is true the data is illustrated in order to make this dependency visible. The graph in Figure

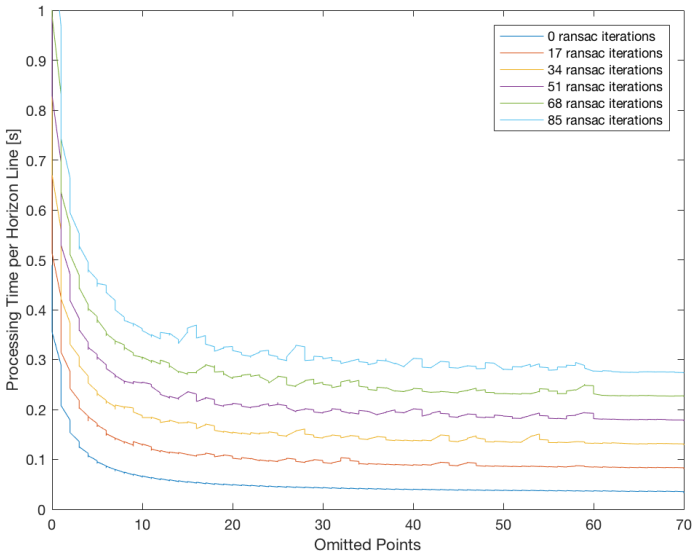


Figure 4.16: Processing Time vs. Omitted Points

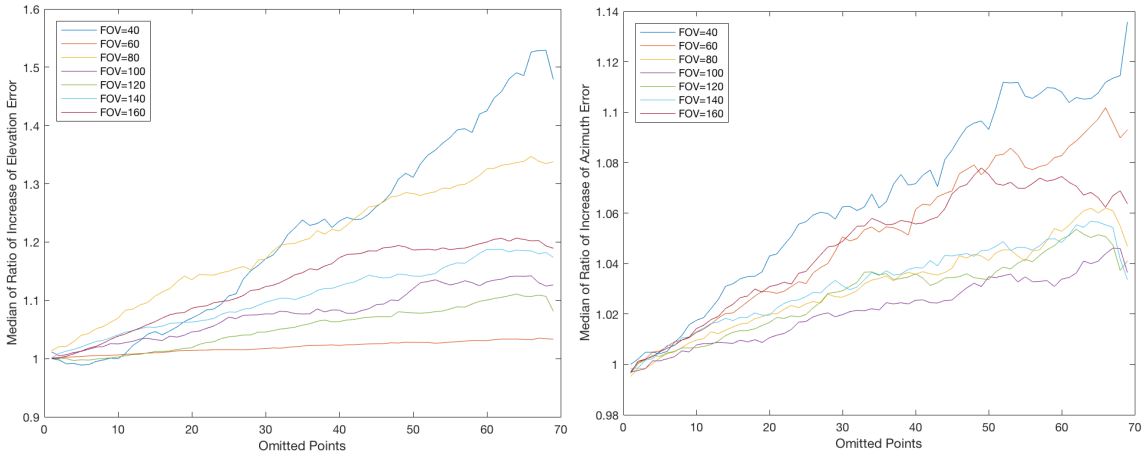
4.16 is a plot of the processing time versus the number of omitted points for multiple numbers of RANSAC iterations. It can be observed, that the curves indeed follow the predicted form. Hence, omitting points is a very effective method to reduce the processing time.

4.5.2 Accuracy

For the design of a optical system that delivers image data for the HorizonSensor it is of importance what kind of effect certain parameters have on the accuracy of the system.

Omitted Points

Omitting points reduces the processing time greatly but also removes measurement points from the calculation and thus will have an effect on the accuracy. Reducing samples is considered to have a negative effect on the accuracy. To determine the effect a series of measurements has been conducted with simulated image data. A total of thirteen videos with 900 frames each were processed with different numbers of omitted points. Each video was created with a different FOV between 40° and 160° at a simulated altitude of 100km. The ratio of the increase of the error in azimuth and elevation for each number of omitted points was recorded. The results are illustrated in Figure 4.17. It can be seen that the median of the ratio between the error with



(a) Elevation Error

(b) Azimuth Error

Figure 4.17: Error Increase vs. Omitted Points

and without omitted points is increasing with the number of omitted points. The elevation error however increases much faster than the azimuth error. The decrease of accuracy with a greater number of omitted points is considered moderate but must be taken into account when using this option. Omitting 10 points is found to be a reasonable trade-off between accuracy (cf. Sec. 4.5.1) and processing time.

Resolution

The effect of the resolution on the accuracy is also a major design factor. Thus, an experiment to determine the influence of the size of the image on the accuracy was conducted. Figure 4.18 shows a graph with a plotted mean and median values of both the azimuth and elevation error for a simulated altitude of 100km. Again,

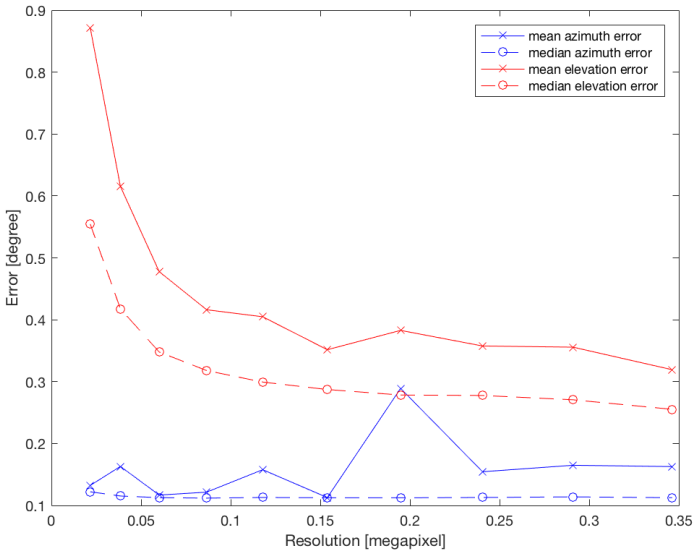


Figure 4.18: Error vs. Resolution

the azimuth error is much lower than the elevation error. Furthermore, the elevation error decreases with increasing resolution whereas the azimuth error stays constant. To reduce the elevation error one can use an optical system with higher resolution, which, however, will increase processing time. A reasonable trade-off between error and processing time is a resolution at about 0.2 megapixels.

Altitude

In contrast to the resolution or the number of omitted points, the altitude is not a design choice. Nevertheless, it is of importance to know how the sensor changes its accuracy within its range of application. Therefore, an experiment to determine the accuracy of the sensor with an image resolution of 780×444 pixels in a range of altitudes. In Figure 4.19 the accuracy error is plotted for multiple FOVs. It is clearly observable, that the azimuth error increases with rising altitude for every FOV but for smaller FOV slower than for greater FOV. The elevation error appear to grow in two modes depending on the FOV and is therefore shown in two graphs (cf. Fig. 4.20).

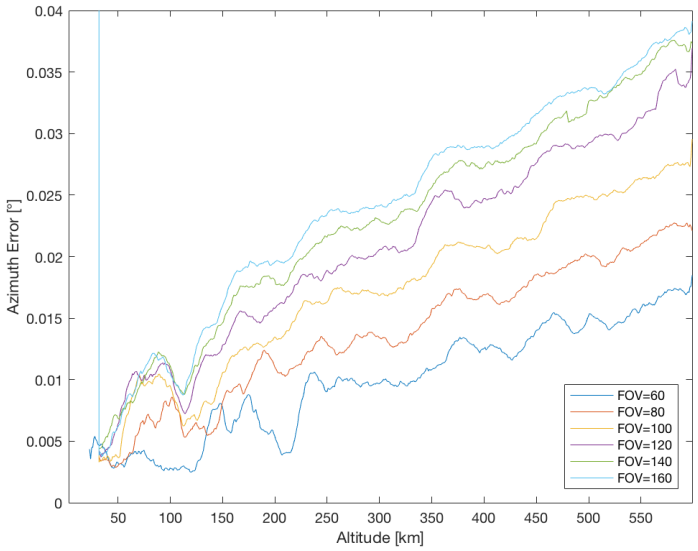
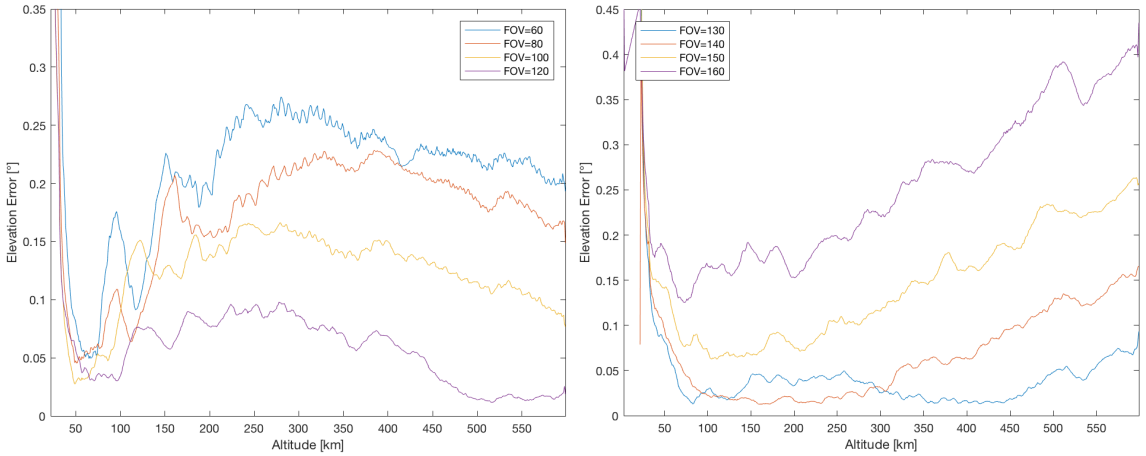


Figure 4.19: Azimuth Error vs. Altitude

The elevation accuracy is, like in all other measurements before, about one order of



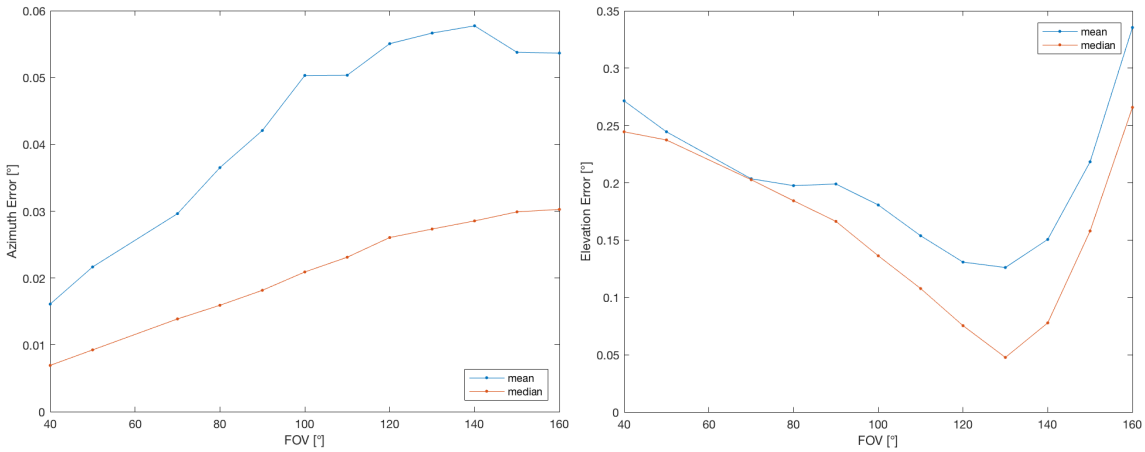
(a) Mode One (b) Mode Two

Figure 4.20: Elevation Error Median vs. Altitude

magnitude worse than the azimuth error. The first mode happens for FOVs between 60° and 120° and the second mode appears for FOVs between 130° and 160°. The first mode starts to rise at about 100km and reaches a maximum at around 300km and starts falling thereafter. The second mode also starts to rise at about 100km but does not reach a maximum but keeps climbing.

Field Of View

Form the analysis performed above one can already get a clue about the influence of the FOV on the accuracy. But since it is a major design parameter, it is here analyzed separately. For a simulated altitude of 100km the azimuth and elevation errors of a simulated optical system with FOVs between 40° and 130° are measured and illustrated in Figure 4.21. In consistency with the analysis above, the elevation



(a) Azimuth Error (b) Elevation Error

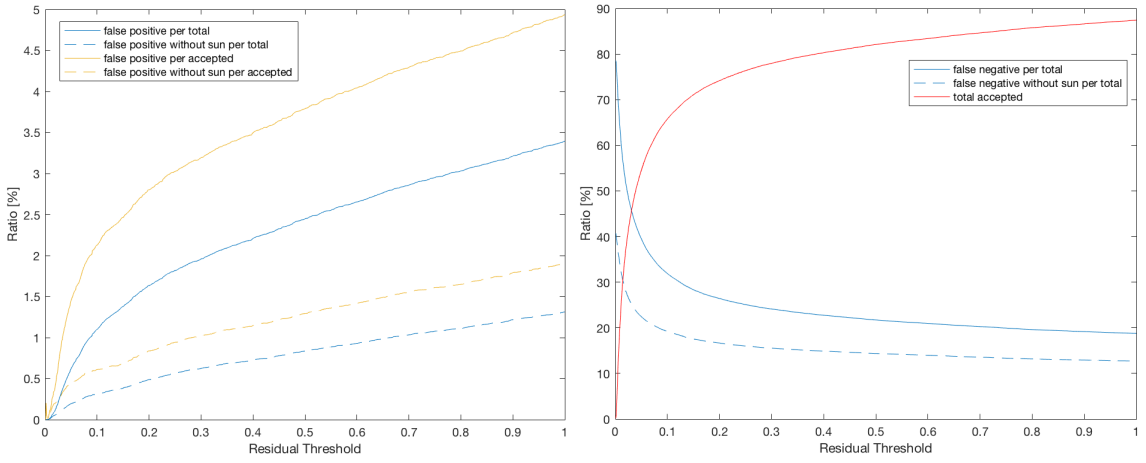
Figure 4.21: Error vs. FOV

error here is again about an order of magnitude worse than the azimuth error. Furthermore, the azimuth error increases with rising FOVs whereas the elevation error decreases down to a minimum at 130° and then increases very fast. Thus a system with a 130° FOV is most accurate in elevation but has an higher error in azimuth. But since the azimuth error is much smaller it is still beneficial to choose an optical system with a FOV of 130°.

4.5.3 Robustness

The robustness can be measured with false positive and false negative errors. False positive errors or false positives are occur if the the algorithm detects the horizon although there is no horizon visible and false negatives occur if the algorithm does not detect a visible horizon. In order to determine these two figures the test case `RexusTest` was implemented. It applies the HorizonSensor algorithm to the real footage from a REXUS flight. The GoPro cameras constantly change the exposure

time in order avoid over or underexposure. In this application this is very important since the incoming energy in the visible spectrum varies wildly between space and Sun. The downside to this is, that the threshold value constantly changes. This would be no problem if the exposure settings of the camera are available, but for this set of data they are not. In order to find the best threshold value the algorithm is applied multiple times with different threshold values to one frame and the result with the smallest residual is selected. This workaround is not necessary if the threshold value can be determined from the camera settings. This test was performed with every 10^{th} threshold value between 70 and 210. The maximum outlier ratio was set to 70% and the detection probability to 99% which leads to 169 RANSAC iterations. A few examples of successful detections can be found in Appendix B.
With the attitude information determined by this test, the `SunSensorTest` (cf. Sec. 4.5.4) and the IMUs, the attitude was estimated with a data fusion algorithm. The fusion however is not part of this thesis work but the result is available. This continuous attitude information has an estimated uncertainty of 1°. With the paradigm developed in Section 2.11, it was used to determine whether the horizon is in the image and if the Sun is within a 160° FOV. This data enabled an automated detection of false positives and false negatives. The residual is a measure of the result's quality produced by the horizon detection and the decision whether a result is accepted depends on it. The user defines a threshold for the residual and results with higher residuals are discarded. High residual thresholds decrease the probability that a correctly detected horizon is discarded but increase the probability that an incorrectly detected horizon is accepted. Depending on the error resistance of the application the residual threshold is increased or decreased by the user. In oder to comprehend the influence of the residual threshold on the ratio of false negatives/positives, they are plotted versus the residual threshold in Figure 4.22. The blue lines in these graphs



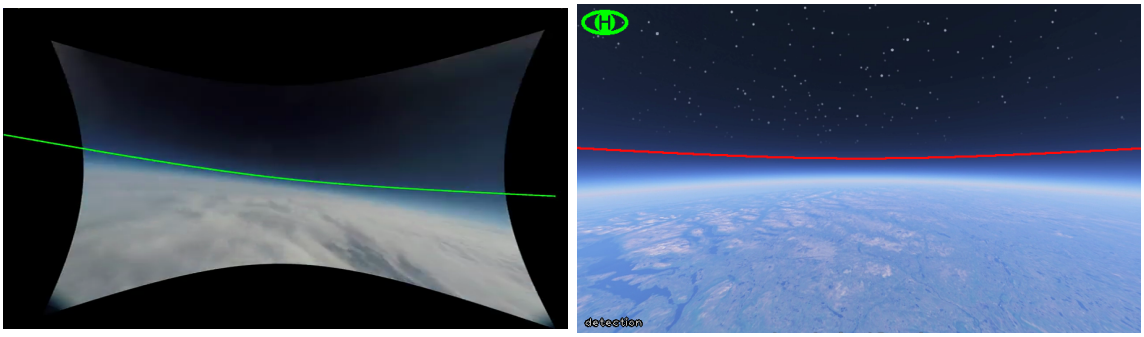
(a) False Positive (b) False Negative

Figure 4.22: Erroneous Detection Ratio vs. Residual

are ratios with the total number of images as reference and the yellow lines take only the sum of images as reference that have not been rejected. Furthermore, for the dashed lines all images with the Sun in the 160° FOV are excluded. The red line is the ratio between accepted and total images.
As predicted the false positives increase and the false negatives decrease with increasing residual threshold. The ratio of false positives is about one order of magnitude smaller than the ratio of false negatives. This is good because for most applications a wrong detection is worse compared to no detection. The ratio of false positives is in any case lower than 5%, for images without the Sun's influence it is even lower than 3%. The residual threshold of 0.2 is a reasonable trade-off between false positives and false negatives. With that threshold, about 75% of the frames are accepted, about 2.5% of the accepted are false positives and about 25% are false negatives.

Minimum Altitude

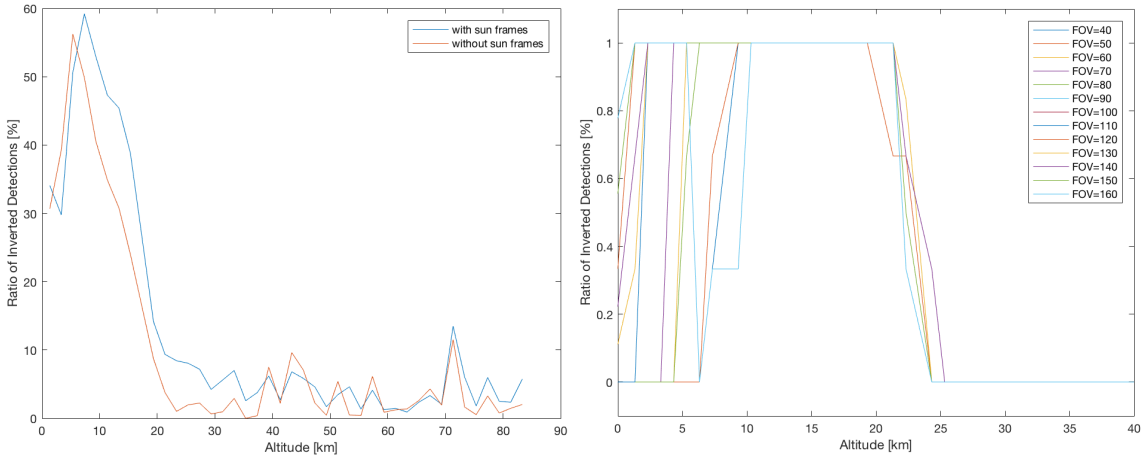
The atmosphere is more visible in lower altitudes and thus has greater influence on the detection than in higher altitudes. The major effect in lower altitudes is that it inverts the curve of the horizon in the binary image.



(a) Real Footage (b) Simulated Footage (GEP)

Figure 4.23: Inverted Detection

The reason for that is not fully understood, but it appears, that for some reason, the brightness of the atmosphere is higher at the borders of the image than in the center. Due to that, the detection is inverted as it can be seen in Figure 4.23. This effect however disappears at a certain altitude, the minimum altitude. In order to determine the minimum altitude the ratio of inverted detections is determined for 2km wide windows and plotted versus the altitude (cf. Fig. 4.24). For both, simulated and real image data, there is a sudden drop in inverted detections at an altitude of around 25km for all FOVs. This drop can also be observed in Figure 4.20a where



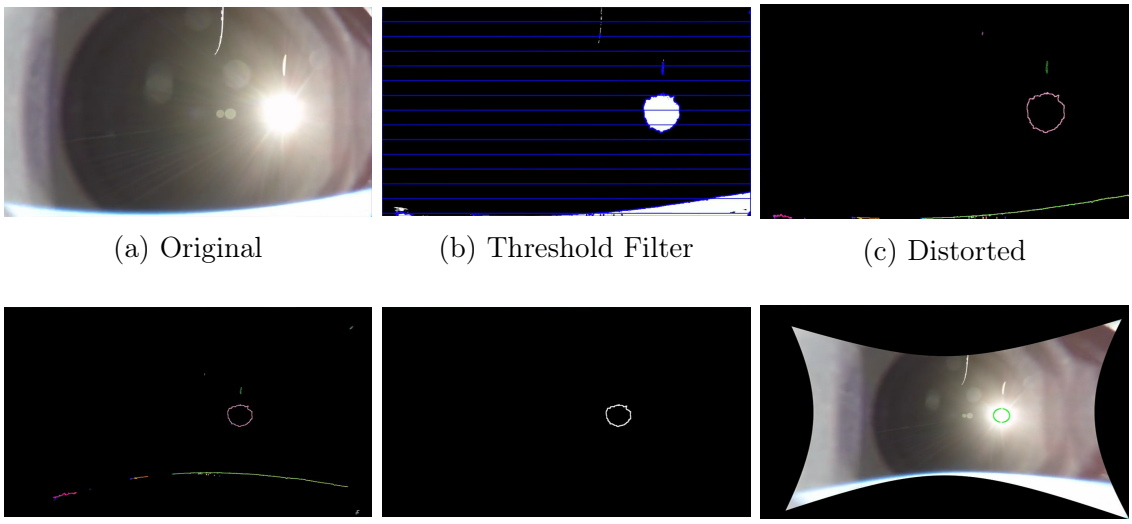
(a) Real Footage (b) Simulated Footage

Figure 4.24: Minimum Altitude

the elevation error suddenly decreases at around 25km. The fact that this effect is observable in both simulated and real image data suggests, that it is neither an simulation artefact nor an measurement error. The HorizonSensor should therefore only be used at altitudes higher than $25km$.

4.5.4 Sun Sensor

During the development of the algorithm the idea arose to test whether it was possible to let the HorizonSensor detect the Sun in the image instead of the Earth. For that purpose a new test case in the HorizonSensorTest environment was implemented and applied to the footage from the REXUS flight. The procedure of the test is very similar to the one used for the horizon detection, only the threshold filter value is constant. Since the Sun is by far the brightest object in the image a very high threshold value is necessary. The value 254 (maximum: 255) turned out to be reasonable. As radius of the central body, the value $15r_s \approx 10^{12}$m, which is approximately 15 times the actual radius r_s of the Sun, was used. The thickness of the atmosphere was chosen to be 10^8m and the altitude was set to 15×10^{10}m which is approximately one astronomical unit (AU). These figures are far from accurate but are applicable to this test. The results are surprisingly positive. An illustration of all steps of a typical example can be seen in Figure 4.25f. After the threshold filter the Sun appears as solid ellipse in the image. The grid (blue) is very small because the expected ellipse is small. Although, a partial horizon is visible after the threshold filter, the RANSAC algorithm is able to find the line that corresponds to the Sun horizon. With that, the azimuth and the elevation can be calculated. Since this test is not an objective of this thesis, no further analysis has been performed. The results, especially those



(a) Original (b) Threshold Filter (c) Distorted

(d) Undistorted (e) RANSAC (f) Result

Figure 4.25: Example of the Sun Detection

between launch and just before the rocket is despun, however, are accurate enough to be used for attitude determination. The data has been used for sensor fusion, alongside the horizon sensor measurements and the IMU, in oder to determine the attitude information described in Section 4.5.3 .

CHAPTER 5

Outlook and Conclusion

5.1 Further Development

Of course, the development of the sensor is not fully completed since the optical system is not selected. Therefore, recommendations towards the design of a future sensor, including the optical system are given. Thereafter, suggestions for improvement of the sensor in a future work are listed.

5.1.1 Design of Future Sensor

The optical system of a future sensor needs to have a high dynamic range and a fast exposure reaction time that is similar to the GoPro cameras. A colored picture is not necessary and rather counter productive because it increases the transmission time of the image. The HorizonSensor only requires monochromatic image data. The camera should transmit settings, like the exposure time, of each image in order to determine the threshold value. A resolution of 0.2 megapixel is found to be a reasonable trade-off between accuracy and sample time. This resolution corresponds to an image size of 448×448 pixels. The optimal FOV of the optical system was found to be 130°. The number of omitted points in the HorizonSensor should be set to 10 since it reduces the processing time significantly while decreasing the accuracy only slightly. In order to reach a detection probability of 95% for data with up to 50% outliers, 23 RANSAC iterations are required. With this setting the processing time on the embedded system is, according to the analysis in Section 4.5.1, about 0.2s. In order to reach a sample rate of 1Hz the transmission of the image may take 0.8s and thus an interface to the camera with a transfer rate of at least 2 Mbit/s is required (assuming an 8bit grayscale resolution). With these settings the magnitude

of the systematic error does not exceed 0.5°, according to the analysis in Section 4.5.2

5.1.2 Improvements

The weak spot of the algorithm is the threshold filter. As mentioned before, every image requires a different threshold value which is dependent on the camera settings. Therefore, an algorithm should be developed to determine the threshold value from the camera setting of an image.
Furthermore, the algebraic fit introduced in Section 2.9 has three drawbacks. Firstly, the minimized algebraic distance, is not the Euclidian distance. Secondly, it is not possible to consider only one branch of an hyperbola. And lastly, several authors mention that there is a systematic offset of algebraic least-squares fits. A fitting algorithm that was able to minimize the Euclidian distance to one branch would be the best solution to the problem. The x-distance (cf. Sec. 2.10) is just a workaround to avoid the high computational intensity of a geometrical distance determination. This workaround only works for very flat hyperbolas and thus might not be suitable for high FOV cameras. A computational effective way to determine the Euclidian distance of a point to a conic would improve the residual determination significantly. In [Zhang, 1996] a method to perform geometrical fitting for conic sections is introduced that could be optimized for this application.
Rensburg [2008] performs a sub-pixel edge detection and improves the accuracy of the results significantly. The same process might improve the results of this application as well.
The implementation of the HorizonSensor on the embedded system could be further optimized by using highly efficient functionalities of the DSP. This could further decrease the processing time.

5.2 Conclusion

Within this thesis work, a robust and flexible software was developed that is able to determine two-axis attitude information from pictures of the Earth horizon in the visible spectrum. The software was implemented for the DSP BlackFin 561 which is part of the onboard computer of sounding rockets launched by the MORABA. The software is based on a new mathematical concept that describes the curve of the horizon in the image as a general conic section. With tests, developed within this thesis, the accuracy, sample time and robustness of the software was measured. It was found, that the systematic error is below 0.5° and that the processing time is between 0.2s and 1s depending on the chosen parameters. The minimum altitude at which the software provides reliable data is 25km. The accuracy of the azimuth

angle was discovered to be roughly one order of magnitude better than the elevation angle accuracy. The sensor does not need a priori information about the location of the horizon in the image and can cope with disturbances introduced by the Sun. These characteristics were determined with simulated data from Google Earth Pro and with real footage of GoPro cameras that were flown on a sounding rocket. The software is able to compensate distortions of the image introduced by fisheye lenses if the intrinsic parameters of the optical system are known. In the scope of this thesis, a tool to determine the intrinsic parameters of a camera was developed. In order to validate the implementation on the embedded system and to measure its performance, a tool for desktop computers was implemented, that simulates an ethernet camera and sends simulated image data to the embedded system. All requirements were met, hence the next step is to implement a camera with the given requirements into the system and test it continuously on sounding rockets or cubesats.

APPENDIX A

Failed Vector Estimation Approaches

A.1 First Approach

A.1.1 Idea

The basic idea of this approach is to first determine the parameters A, B, C, D, E and F with the least-squares method and then use those parameters to determine the attitude. The output would then be a three-dimensional line-of-sight vector from the camera origin to the center of the Earth.

A.1.2 Least-Square Problem Statement

Let (x_i, y_i) be the coordinate of the point i and n be the number of points in the horizon line. The equation of the conic section for every point is

$$h_i R = 0 \tag{A.1}$$

where

$$h_i = \begin{pmatrix} x_i^2 & x_i y_i & y_i^2 & x_i & y_i & 1 \end{pmatrix} \tag{A.2}$$

and

$$R = \begin{pmatrix} A & B & C & D & E & F \end{pmatrix}^T. \tag{A.3}$$

This is an over-determined system of equations with six unknown and n equations which can be solved with the Least-Square method. The most simple solution of this system is the trivial solution $\begin{bmatrix}0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T$. In order to avoid the trivial solution the solution must be constrained. There are linear and quadratic constraints introduced in the literature. However, quadratic constraints require the calculation of the eigenvectors. This computational intensiv and shall be avoided. Besides, tests showed that the results with these quadratic constraints are not significantly better than with linear constraints. Rosin introduced two linear constraints:

$$F = 1 \tag{A.4}$$

$$A + C = 1 \tag{A.5}$$

Constraining the Least-Square algorithm to one of these constrains prevents the solution from becoming the trivial solution. Least-Square ansatz can be written as

$$R = (H^T H)^{-1} V^T (V^T (H^T H)^{-1} V)^{-1} \tag{A.6}$$

where

$$H = \begin{bmatrix}h_1 & h_2 & h_3 & \dots & h_n\end{bmatrix}^T \tag{A.7}$$

and V shall be replaced by one of the constraint vectors

$$V_1 := \begin{pmatrix}0 & 0 & 0 & 0 & 0 & 1\end{pmatrix} \quad (F = 1) \tag{A.8}$$

$$V_2 := \begin{pmatrix}1 & 0 & 1 & 0 & 0 & 0\end{pmatrix} \quad (A + C = 1). \tag{A.9}$$

Since $(V(H^T H)^{-1} V^T)^{-1}$ is a scalar and the multiplication of an equation with a scalar does not change the equation, this may be ignored. Using that and the substitution $S := H^T H$ the equation A.6 can be simplified to

$$R_1 = S^{-1} V_1^T \tag{A.10}$$

$$R_2 = S^{-1} V_2^T. \tag{A.11}$$

From these two results, the one with the smallest residual error is chosen. The matrix

$$S := \begin{bmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} & s_{16} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} & s_{26} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} & s_{36} \\ s_{41} & s_{42} & s_{43} & s_{44} & s_{45} & s_{46} \\ s_{51} & s_{52} & s_{53} & s_{54} & s_{55} & s_{56} \\ s_{61} & s_{62} & s_{63} & s_{64} & s_{65} & s_{66} \end{bmatrix} \tag{A.12}$$

is a symmetrical 6×6 matrix where each component is some sum over the whole horizon line. The equations in A.13 define one half of the components and the other half is given through the symmetry $s_{ij} = s_{ji} \quad \forall i, j \in [1, 6]$. To achieve a clearer view,

the limits of each sum and the indexes of x and y are omitted but it is always meant $\sum_{i=1}^{n}$ and x_i or y_i.

$$
\begin{array}{lllllllllll}
s_{11} = & \sum x^4 & s_{22} = & s_{31} & s_{33} = & \sum y^4 & s_{44} = & s_{51} & s_{55} = & s_{63} \\
s_{21} = & \sum x^3y & s_{32} = & \sum xy^3 & s_{43} = & s_{52} & s_{54} = & s_{62} & s_{65} = & \sum y \\
s_{31} = & \sum x^2y^2 & s_{42} = & s_{51} & s_{53} = & \sum y^3 & s_{64} = & \sum x & s_{66} = & n \\
s_{41} = & \sum x^3 & s_{52} = & \sum xy^2 & s_{63} = & \sum y^2 & & & & \\
s_{51} = & \sum x^2y & s_{62} = & \sum xy & & & & & & \\
s_{61} = & \sum x^2 & & & & & & & &
\end{array}
$$

A.1.3 Vector Calculation

The Equations 2.22 -2.27 can also be expressed as

$$A = r_e^2 - e_{c,y}^2 - e_{c,z}^2 \tag{A.13}$$

$$B = e_{c,x}e_{c,y} \tag{A.14}$$

$$C = r_e^2 - e_{c,x}^2 - e_{c,z}^2 \tag{A.15}$$

$$D = -e_{c,x}e_{c,z}f \tag{A.16}$$

$$E = -e_{c,y}e_{c,z}f \tag{A.17}$$

$$F = -f^2(e_{c,x}^2 + e_{c,y}^2 - r_e^2) \tag{A.18}$$

where $\vec{e_c}$ is the vector from camera origin to the center of the Earth. Rearanging each line gives

$$e_{c,y}^2 + e_{c,z}^2 = r_e^2 - A \tag{A.19}$$

$$e_{c,x}^2 + e_{c,y}^2 = r_e^2 - \frac{F}{f^2} \tag{A.20}$$

$$e_{c,x}^2 + e_{c,z}^2 = r_e^2 - C \tag{A.21}$$

$$e_{c,x}e_{c,y} = B \tag{A.22}$$

$$e_{c,x}e_{c,z} = -\frac{D}{f} \tag{A.23}$$

$$e_{c,y}e_{c,z} = -\frac{E}{f}. \tag{A.24}$$

The equations

$$e_{c,x} = \pm\sqrt{\frac{1}{2}\left(r_e^2 - C + A - \frac{F}{f^2}\right)} \quad \text{for} \quad r_e^2 + A \geq C + \frac{F}{f^2} \tag{A.25}$$

$$e_{c,y} = \pm\sqrt{\frac{1}{2}\left(r_e^2 + C - A - \frac{F}{f^2}\right)} \quad \text{for} \quad r_e^2 + C \geq A + \frac{F}{f^2} \tag{A.26}$$

$$e_{c,z} = \pm\sqrt{\frac{1}{2}\left(r_e^2 - C - A + \frac{F}{f^2}\right)} \quad \text{for} \quad r_e^2 + \frac{F}{f^2} \geq A + C \tag{A.27}$$

can be derived by (A.21)+(A.20)-(A.19)=(A.25), (A.19)+(A.20)-(A.21)=(A.26) and (A.19)+(A.21)-(A.20)=(A.27). The equations A.25 - A.27 represent eight solutions but only three of the six parameters were used so far. The remaining three parameters can be used to determine the signs. With the literals

$$b := B \geq 0 \tag{A.28}$$

$$d := D \geq 0 \tag{A.29}$$

$$e := E \geq 0 \tag{A.30}$$

and some logical calculation it can be derived that the vector from the camera origin to the center of the Earth is given by

$$\vec{e}_c^{\pm} = \begin{cases} \pm \begin{bmatrix} -|e_{c,x}| & |e_{c,y}| & -|e_{c,z}| \end{bmatrix}^T & \text{for } bd\bar{e} \vee \bar{b}\bar{d}e \\ \pm \begin{bmatrix} -|e_{c,x}| & |e_{c,y}| & |e_{c,z}| \end{bmatrix}^T & \text{for } b\bar{d}e \vee \bar{b}d\bar{e} \\ \pm \begin{bmatrix} |e_{c,x}| & |e_{c,y}| & |e_{c,z}| \end{bmatrix}^T & \text{for } \bar{b}de \vee b\bar{d}\bar{e} \\ \pm \begin{bmatrix} |e_{c,x}| & |e_{c,y}| & -|e_{c,z}| \end{bmatrix}^T & \text{for } \bar{b}\bar{d}\bar{e} \vee bde. \end{cases} \tag{A.31}$$

The vector now only has two solutions, the correct one and its oposite direction.

A.1.4 Problem

The resulting conic section aligned perfectly with the data points but when comparing the calculated vector with the real one they did not match, not even approximately. The vector was off by several magnitudes and the direction was not even close to where it should be. The problem seems to be, that there is an infinite amount of parametrizations for one and the same conic section. One of the transformations of the parameters without changing the conic section is the scaling of all parameters with same arbitrary, non-zero factor. The fitting algorithm allows more parametrizations than those that fullfill the Equations A.19 - A.20.

A.2 Second Approach

A.2.1 Idea

To address the issue of the last approach and only allow solutions that fullfill Equations A.19 - A.20, the least square method was constrained more strongly. By eliminating $e_{c,x}$, $e_{c,y}$ and $e_{c,z}$ in A.19 - A.20 two sets of three new equations can be derived. One with at most quadratic terms

$$0 = r_e^4 - \frac{2r_e^2F}{f^2} - C^2 + 2AC - A^2 + \frac{F^2}{f^4} - 4B^2 \tag{A.32}$$

$$0 = r_e^4 - 2Cr_e^2 + C^2 - A^2 + \frac{2AF}{f^2} - \frac{F^2}{f^4} - \frac{4D^2}{f^2} \tag{A.33}$$

$$0 = r_e^4 - \frac{4E^2}{f^2} - 2Ar_e^2 + \frac{2CF}{f^2} + A^2 - C^2 - \frac{F^2}{f^4} \tag{A.34}$$

and one with at most cubic terms

$$0 = ADBf^2 - DBf^2r_e^2 + B^2Ef^2 + D^2E \quad (A.35)$$

$$0 = CEBf^2 - EBf^2r_e^2 + B^2Df^2 + DE^2 \quad (A.36)$$

$$0 = r_e^2ED - \frac{EFD}{f^2} - BD^2 - BE^2. \quad (A.37)$$

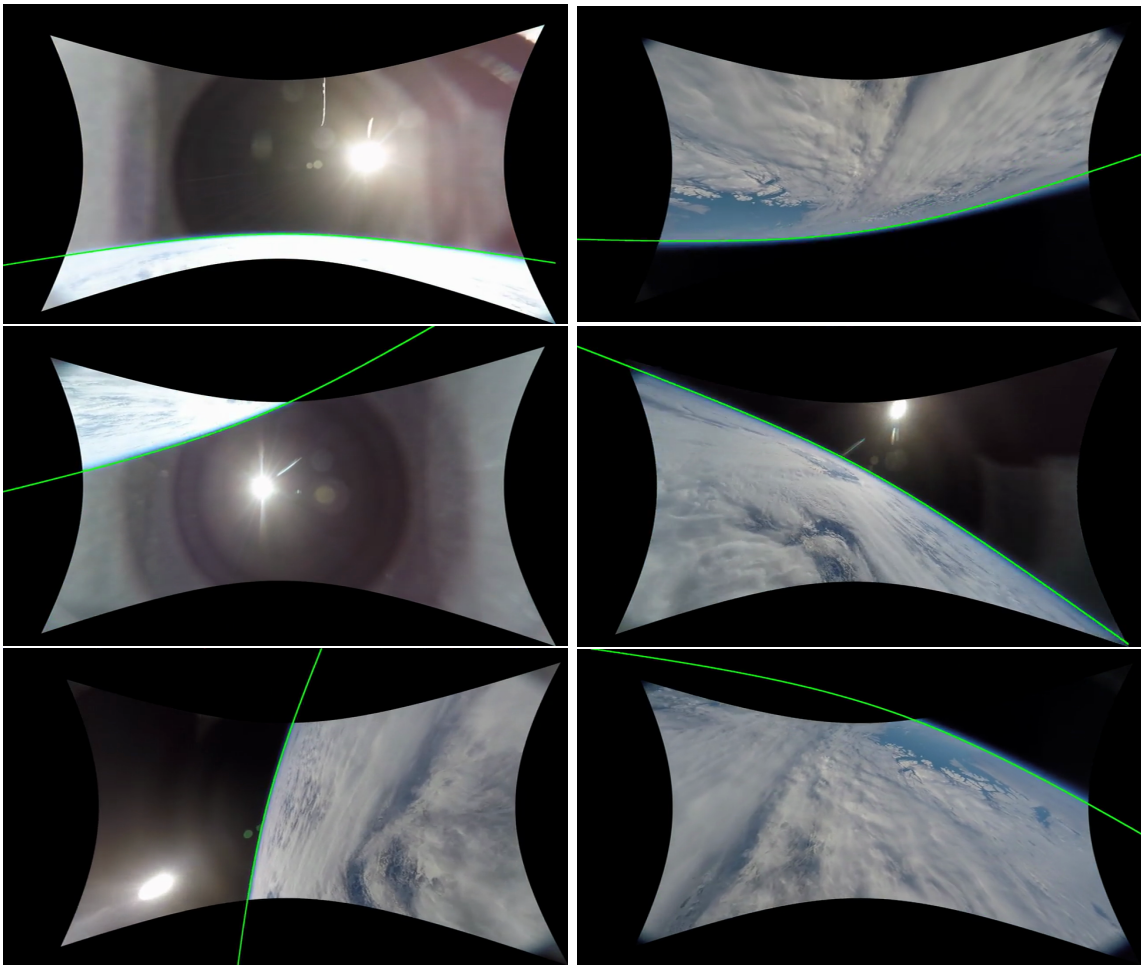
The parameters must fullfill these equations in order satisfy Equations A.19 - A.20 and thus make a statement about the attitude possible.
For quadratic and cubic constraints there is no closed form for the least-squares method hence an iterative least-squares procedure is required. The implementation of this method is quite time-consuming, therefore the `fmincon` function of Matlab's optimization toolchain was used to proof the concept.

A.2.2 Problem

The problem with this procedure is that it also finds local minima. Unfortunately, there seem to be many local minima within this problem and it is not always possible to give a good start solution. Although the constraints above were met in all solutions found by the procedure, the calculation of the attitude was not possible, because most of the time the algorithm found some local minimum that was far away from the global one.

APPENDIX B

Horizon Detection Examples



List of Abbreviations

ADCS attitude determination and control system.

AOS acquisition of signal.

AU astronomical unit.

CMOS complementary metal-oxide-semiconductor.

COTS commercial off-the-shelf.

DLR German Aerospace Center.

DMARS Digital Miniature Attitude Reference System.

DSP digital signal processor.

ECEF Earth centered Earth Fixed.

ENU east-north-up.

FOV field of view.

GEO geostationary orbit.

GEP Google Earth Pro.

GSOC German Space Operation Center.

GUI graphical user interface.

HORACE Horizon Acquisition Experiment.

HS horizon sensor.

IDE integrated development environment.

IMU inertial measurement unit.

IR infrared.

ITAR International Traffic in Arms Regulations.

KML Keyhole Markup Language.

LEO low Earth orbit.

LMA Levenberg-Marquardt algorithm.

LOS loss of signal.

MFC multi function card.

MORABA Mobile Rocket Base.

MSL mean sea level.

NASA National Aeronautics and Space Administration.

NED north-east-down.

PATHOS Position-Vector Acquisition Through Horizon Observation System.

PCB printed circuit board.

RANSAC random sample consensus.

RFT Space Flight Technology.

RPS revolutions per second.

WGS84 World Geodetic System 1984.

XML Extensible Markup Language.

List of Figures

List of Tables

List of Algorithms

Bibliography

World geodetic system 1984. Technical report, US National Imagery and Mapping Agency, January 2000.

Mohamad Nazree Dol Bahar, Mohd Effandi Mohd Hassan, Norhizam Hamzah, Ahmad Sabirin Arshad, Xandri Farr, Lourens Visagie, and Willem Herman Steyn. Modular cmos horizon sensor for small satellite attitude determination and control subsystem. 2006.

J. Barf, T. Rapp, M. Bergmann, S. Geiger, A. Scharf, and F. Wolz. Horizon Acquisition for Attitude Determination Using Image Processing Algorithms- Results of HORACE on REXUS 16. In Ouwehand L., editor, *22nd ESA Symposium on European Rocket and Balloon Programmes and Related Research*, volume SP-730 of *ESA Special Publication*, pages 627–634, September 2015.

Jochen Barf. Development and Implementation of an Horizon Sensing Algorithm based on Image Processing Technologies. Bachelor Thesis, Julius-Maximilian University of Würzburg, October 2014.

Peter Berlin. *The Geostationary Applications Satellite*. Cambridge Aerospace Series. Cambridge University Press, 1988. ISBN 0 521 33525 6.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009. ISBN 978-0-262-03384-8.

Donald M. Crisman. Horizon detection in the visible spectrum, September 2016.

Cube Space. CubeSense Brochure, 2016. URL `https://www.cubesatshop.com/product/cubesense/`.

Anton H.J. de Ruiter, Christopher J. Damaren, and James R. Forbes. *Spacecraft Dynamics and Control*. Wiley, 2013. ISBN 9781118342367.

EADS Sodern. Earth and attitude sensor STD15/STD16, satellite attitude, attitude control - Sodern. URL http://www.sodern.com/sites/en/ref/Earth-sensors_52.html.

Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL http://doi.acm.org/10.1145/358669.358692.

Andrew W. Fitzgibbon, Maurizio Pilu, and Robert B. Fisher. Direct Least Squares Fitting of Ellipses, January 1996.

GoPro. (FOV) der HERO4, 2017. URL https://de.gopro.com/help/articles/Question_Answer/HERO4-Field-of-View-FOV-Information.

Inertial Science Inc. *Digital Miniature Attitude Reference System DMARS-r*. 3533 Old Conejo Road Suite 104, Newbury Park, U.S.A., 1999.

Juho Kannala and Sami S. Brandt. A generic camera model and calibration method for conventional, wide-angle, and fish-eye lenses. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(8):1335–1340, August 2006. ISSN 0162-8828. doi: 10.1109/TPAMI.2006.153. URL http://dx.doi.org/10.1109/TPAMI.2006.153.

Wilfried Ley, Klaus Wittmann, and Willi Hallmann, editors. *Handbook of Space Technology*. Wiley, 2009. ISBN 978-0-470-69739-9.

Malcolm Macdonald and Viorel Badescu, editors. *The International Handbook of Space Technology*. Springer, 2014. ISBN 978-3-642-41100-7.

Leonardo Mazzini. *Flexible Spacecraft Dynamics, Control and Guidance*. Springer, 2016. ISBN 978-3-319-25538-5.

David Meller, Prapat Sripruetkiat, and Kristin Makovec. Digital cmos cameras for attitude determination. 2000.

Sergio Montenegro and Frank Dannemann. RODOS - Real Time Kernel Design for Dependability, 2009.

MORABA. Mobile Raketenbasis des DLR - Raumflugbetrieb und Astronautentraining. URL http://dlr-gsoc.de/moraba2015/.

Vincent L. Pisacane, editor. *Fundamentals of Space Systems*. Oxford University Press, 2nd edition, 2005. ISBN 978-0-19.516205-9.

Thomas Rapp, Jochen Barf, Matthias Bergmann, Sven Geiger, Arthur Scharf, and Florian Wolz. Horace - Student Experiment Document. October 2014.

Helgard Marais Van Rensburg. An Infrared Earth Horizon Sensor for a LEO Satellite, March 2008.

P.L. Rosin. A note on the least square fitting of ellipses. *Pattern Recognition Letters*, (14):661–699, October 1993.

Sellex Galileo. Coarse InfraRed Earth Sensor IRES-C | ESA's ARTES Programmes. URL `https://artes.esa.int/projects/coarse-infrared-earth-sensor-ires-c`.

Satoshi Suzuki and Keiichi Abe. Topological Structural Analysis of Digitized Binary Images by Border Following. *Computer Vision, Graphics and Image Processing*, 30:32–46, December 1983.

UB-SPACE. UB-SPACE - Image Processing for Determination of relative Satellite Motion. URL `http://www.ub-space.de/`.

Thomas Uhlig, Florian Sellmaier, and Michael Schmidhuber, editors. *Spacecraft Operations*. Springer, 2015. ISBN 978-3-7091-1802-3.

Dominik Wagner, Moritz Aicher, Jonas Ehnle, Elke Heidmann, Bastian Klein, Felix Klesen, Florian Kunzi, Liviu Stamat, and Kevin Chmiela. Pathos - Student Experiment Document. June 2016.

Markus Wittkamp and Anderson Zigiotto. A very High Performance Multi Purpose Computing Card for TM/TC and Control Systems. In *European Test and Telemetry Conference (ETTC'09)*, June 2009. URL `http://elib.dlr.de/63516/`.

Zhengyou Zhang. Parameter estimation techniques: a tutorial with application to conic fitting. *Image and Vision Computing*, 15:59–76, April 1996.

Zhengyou Zhang. A flexible new technique for camera calibration. volume 22, pages 1330–1334, December 2000. URL `https://www.microsoft.com/en-us/research/publication/a-flexible-new-technique-for-camera-calibration/`.